User manual

XperiDo

Last updated: 14-01-2015

XD XperiDo

Table of contents

1 Introduction

1.1 XperiDo Nucleus

This document is the user manual for XperiDo Nucleus (XDNucleus). XperiDo Nucleus is a ready-to-use, server-side component for asynchronous document creation and output management, easy to embed in your existing software applications. It combines a flawless end user experience with Microsoft Word based template design (thanks to the XperiDo add-in for Microsoft Word) and script based customization capabilities.

1.2 About this document

1.2.1 Disclaimer

In this manual, we try our best to explain every function of XDNucleus in a manner that's understandable to everyone. If something isn't clear, wrong or missing, please let us know by sending an email to info@invenso.com.

The images in this document are taken from various versions of XDNucleus. If you notice that the screenshots in this manual don't always fully correspond to your screen, know that the main functionality hasn't changed, and that purely aesthetic changes might not be updated immediately in the manual.

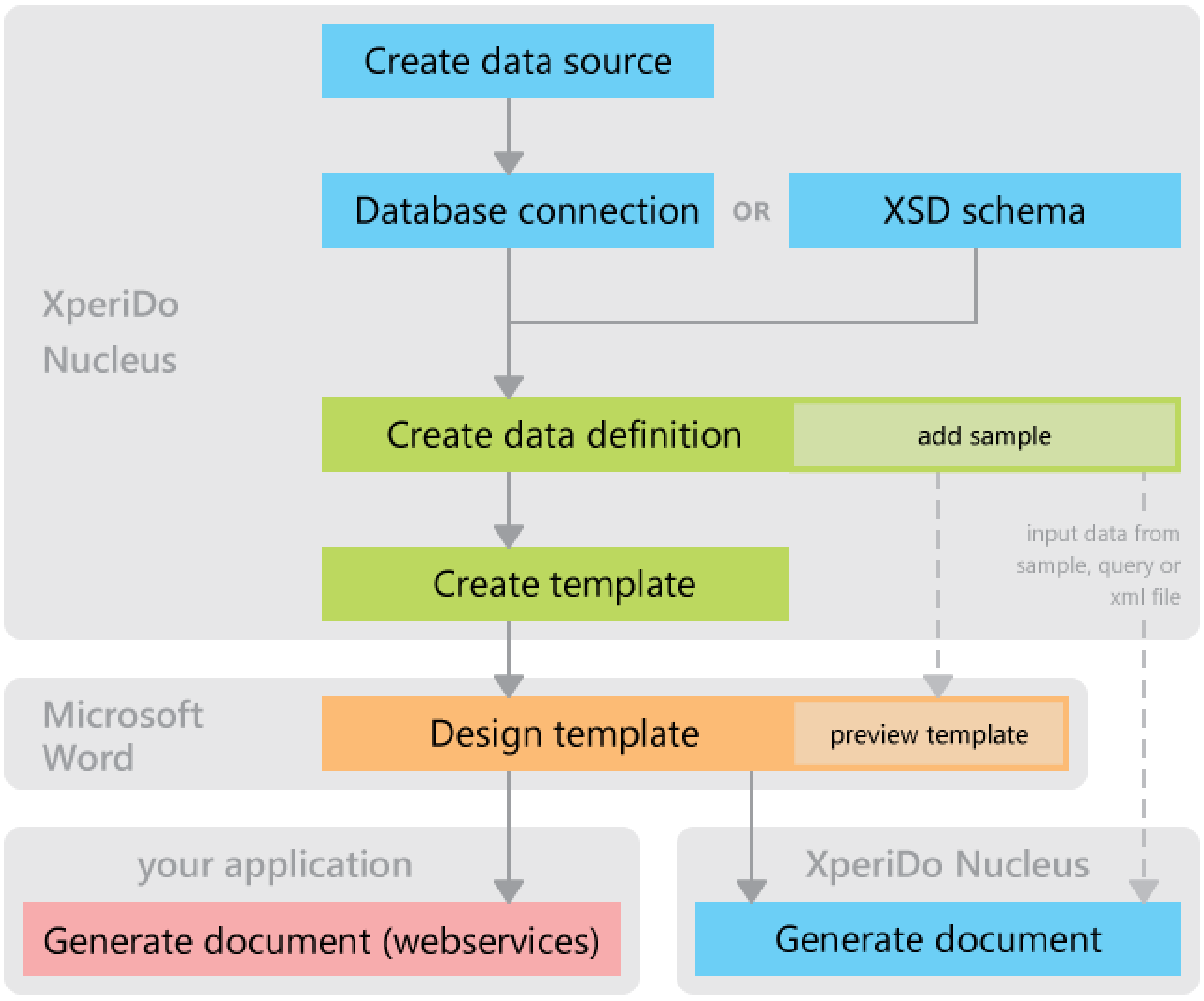
1.2.2 Version

This manual best describes the features found in XDNucleus 2.0.

2 General

2.1 Workflow

Creating documents with XperiDo Nucleus is done by following a certain workflow:

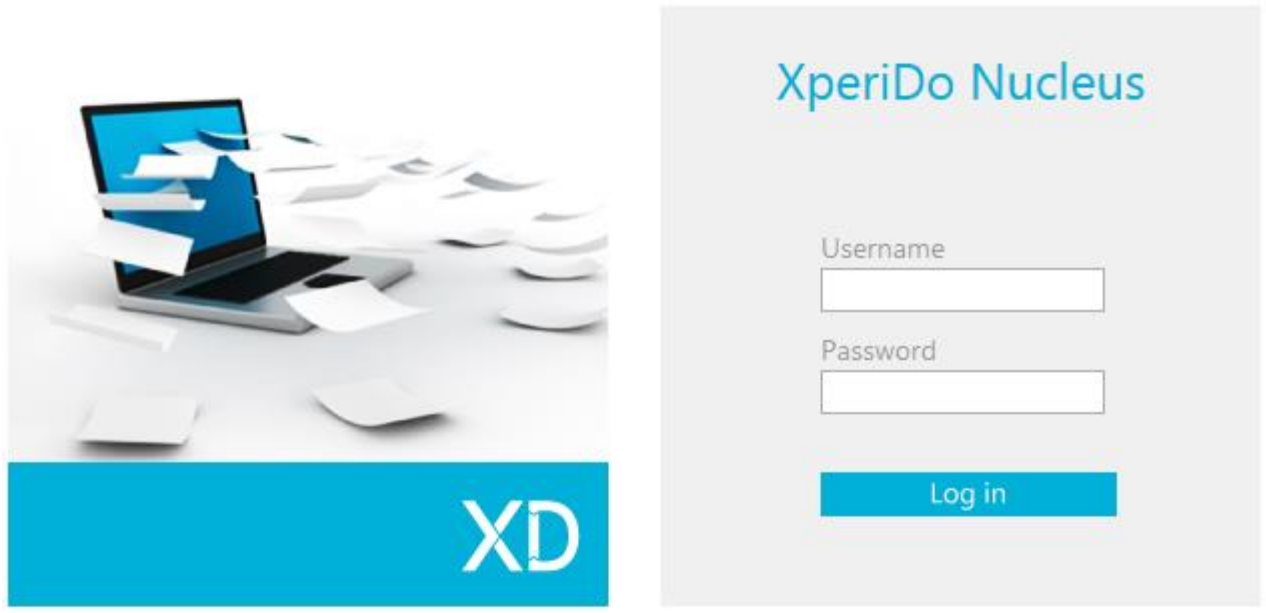


First, you decide where your data comes from. This can be an external database, in which case you create a data source and a database connection. Or, you pull the data from an XSD schema, which you upload. In both cases, you need to create a data definition based on the database/XSD schema. With this data definition, you create a template which you then design in Word by using the XperiDo add-in for Word. If you add a sample to the data definition, you can preview your template in Word.

When your template is defined and designed, you can then generate a document. You can do this in XperiDo Nucleus, by using input data from a sample, a query or an XML file. If you want to create multiple documents, you can use the offered webservices to generate a document in your own application.

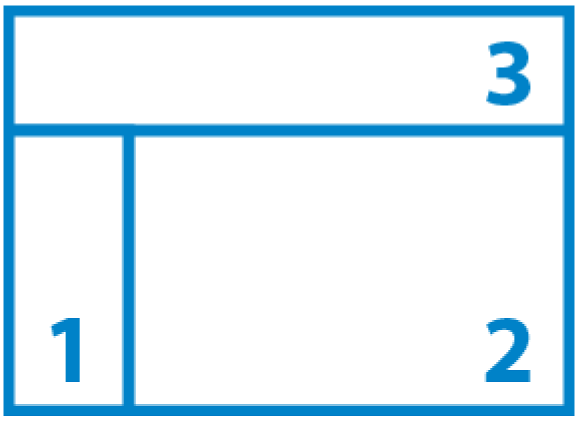
2.2 Logging in

Before you can work with XperiDo Nucleus, you need to log in. Enter your username and password in the text input fields and click Log in.



2.3 Sections

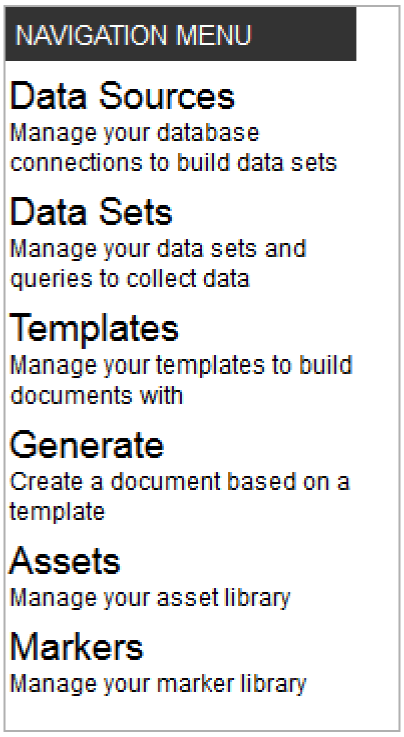
The screen of XperiDo Nucleus consists of different sections:



Section 1 is the navigation menu. This is where you select which area of XperiDo Nucleus you want to navigate to.
Section 2 is the main section.The content of this section depends on the area you selected in the document flow controls section.
Section 3 is the header section. Here you'll find various general options about XperiDo Nucleus.

2.3.1 Navigation menu

Navigating the primary areas of XperiDo Nucleus is done by clicking on an item in the navigation menu , which isalways shown to the left of the main section.


NAVIGATION MENU

Data Sources
Manage your database connections to build data sets

Data Sets
Manage your data sets and queries to collect data

Templates
Manage your templates to build documents with

Generate
Create a document based on a template

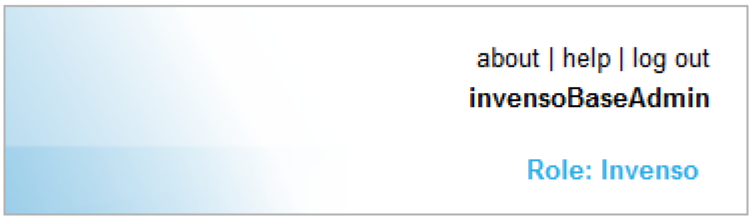
Assets
Manage your asset library

Markers
Manage your marker library

2.3.2 Main

The contents of the main section will change depending on the area you've chosen in the navigation menu. Each area is described in the corresponding section of this manual.

2.3.3 Header



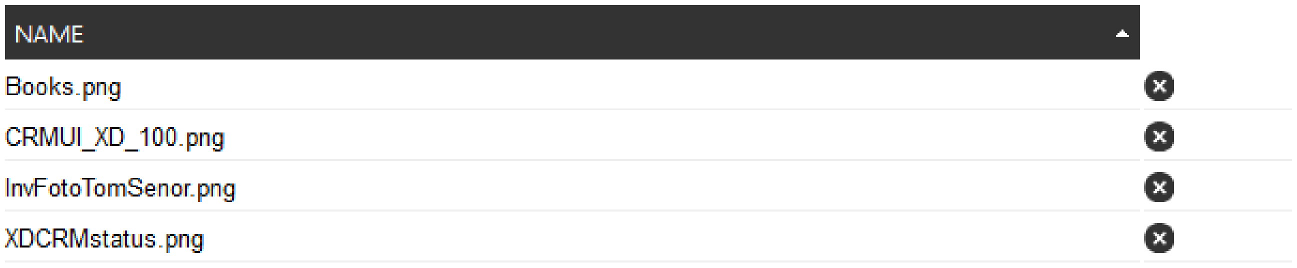
The header, like the navigation menu, is stationary. The header features a few buttons/options:

- Click about to bring up information regarding this version of XperiDo Nucleus.
- Click log out to log out of XperiDo Nucleus.
- Check the current role. Usually there is only one role.
- Click anywhere else to return to the start screen.

2.4 Navigation

Inside the main section, there are different types of elements which are used to navigate XperiDo Nucleus:

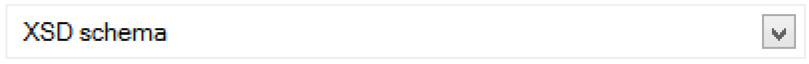
2.4.1 Lists



NAME
Books.png
CRMUI_XD_100.png
InvFotoTomSenor.png
XDCRMstatus.png

Lists allow you to view items by clicking on them. You can delete an item in a list by clicking the X to the right of the item. When deleting, a confirmation message will be shown.

2.4.2 Drop-down lists

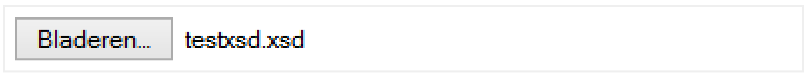


Drop-down lists ask you to choose one of the listed option. They are opened by clicking on the icon to the right. Click on the desired option to select it. The selected option will be shown.

2.4.3 Buttons

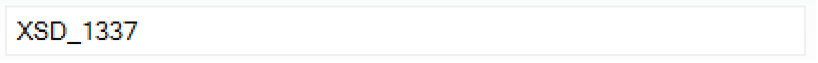
Buttons, when clicked on, trigger an action. Which action it triggers is displayed on the button.

2.4.4 Browse buttons



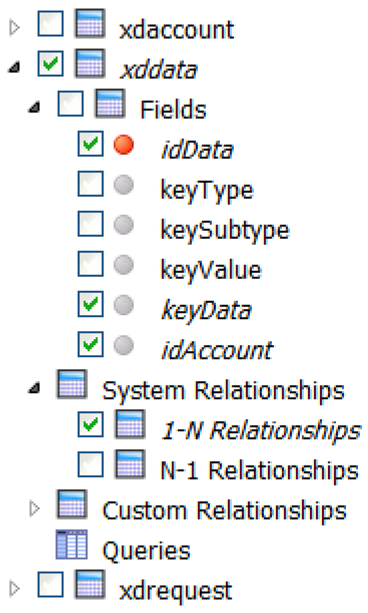
Browse buttons (bladeren in Dutch) ask you for a file. Click the button to open a browse dialog to search for the file you want to upload. The name of the selected file will be displayed next to the button.

2.4.5 Text input fields



Text input fields ask you for text, such as a name or a label. Place the cursor in the field and start typing.

2.4.6 Data selection



The data selection occurs in the data definition area. How to use it is explained in that section of this manual.

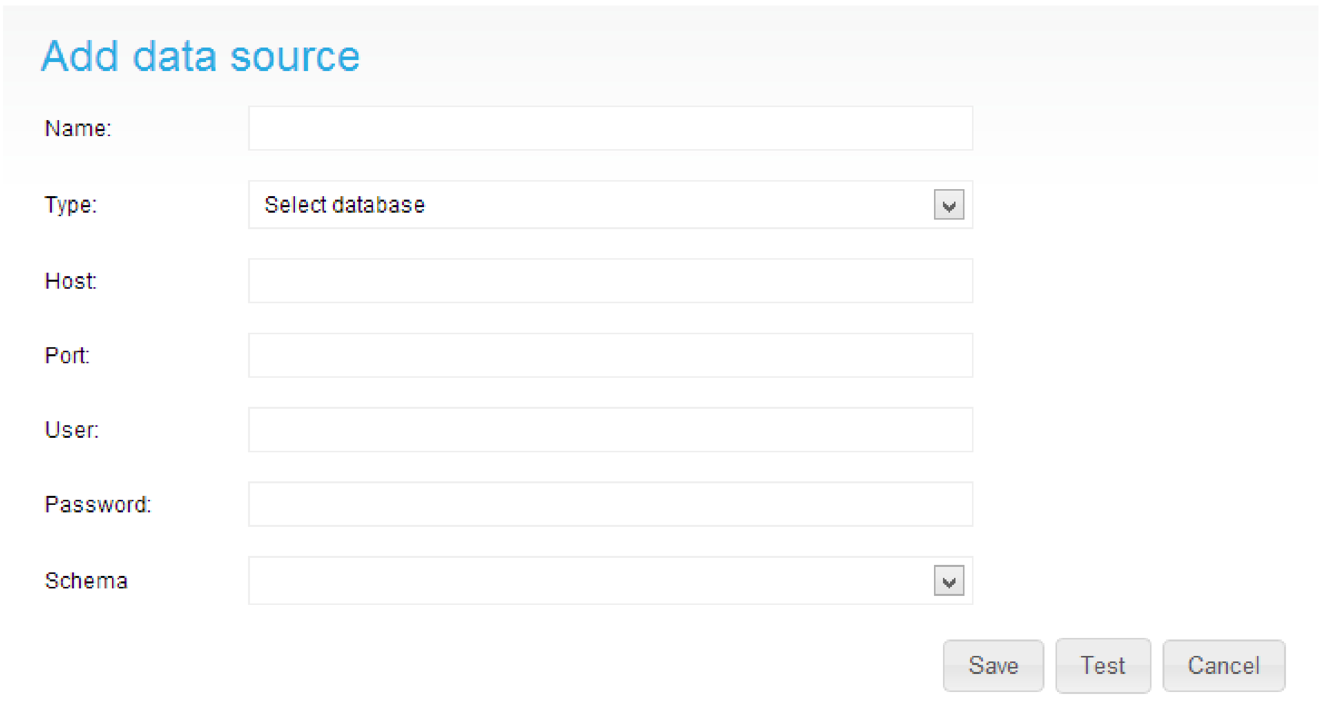
3 Data

XperiDo Nucleus requires data to work. This data can come from either a database or an XSD schema.

3.1 Database (data source)

If you want to use the data inside a database, you need to create a data source. A data source is a connection to a table in a database. Click Data Sources in the document flow controls to bring up a screen where you can create new data sources or edit existing ones.

Clicking New Data Source (or clicking an existing data source) brings up a screen where you can input the properties of the data source:


Add data source

Name:	
Type:	Select database
Host:	
Port:	
User:	
Password:	
Schema	

- Name: the name of the data source.
- Type: the type of database you're connecting to. XperiDo Nucleus supports MySQL, DB2 and MSSQL.
- Host: where the database is hosted.
- Port: the port via which to connect to the database.
- User: the user used to log in to the database.
- Password: the password associated with the user.
- Schema: the default table you want to use in the data source (this can be changed later). If the connection isn't valid, you can't choose a table.

Click Test to test the connection, click Cancel to stop the creation/editing and click Save to save the data source.

3.2 XSD schema

An XSD schema is a file that specifies how to describe the elements in an XML document. Put simply, an XSD schema doesn't contain any data, it merely specifies how data should be described. XSD schemas are uploaded during the data definition creation process.

The following is an example of an XSD schema: (examples from http://www.w3schools.com/schema/schema_example.asp)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

This XSD schema defines an element shiporder with child elements orderperson, shipto and item.

The following is an example of an XML file that is valid against the above schema:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

For more information about XSD schemas, visit http://en.wikipedia.org/wiki/XML_Schema_%28W3C%29.

4 Data definitions

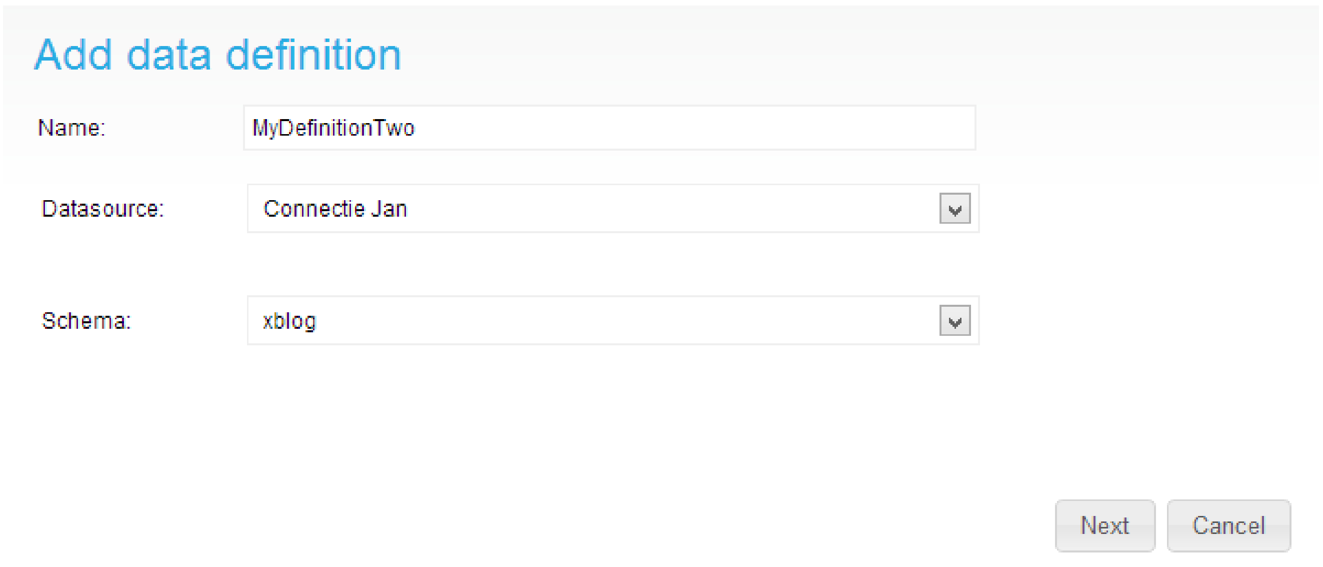
Data definitions specify which data is to be used in a template, with zero or more samples added which can be used for previewing the template or the creation of a document inside XperiDo Nucleus. Click Data Definitions in the document flow controls to bring up a list of all the current data definitions.

4.1 Creating a data definition

Click New Data Definition. In this screen you can give your data definition a name and select a datasource. In this context, datasource means either a database connection or an XSD schema. The drop-down list lists all data sources as well as XSD schema.

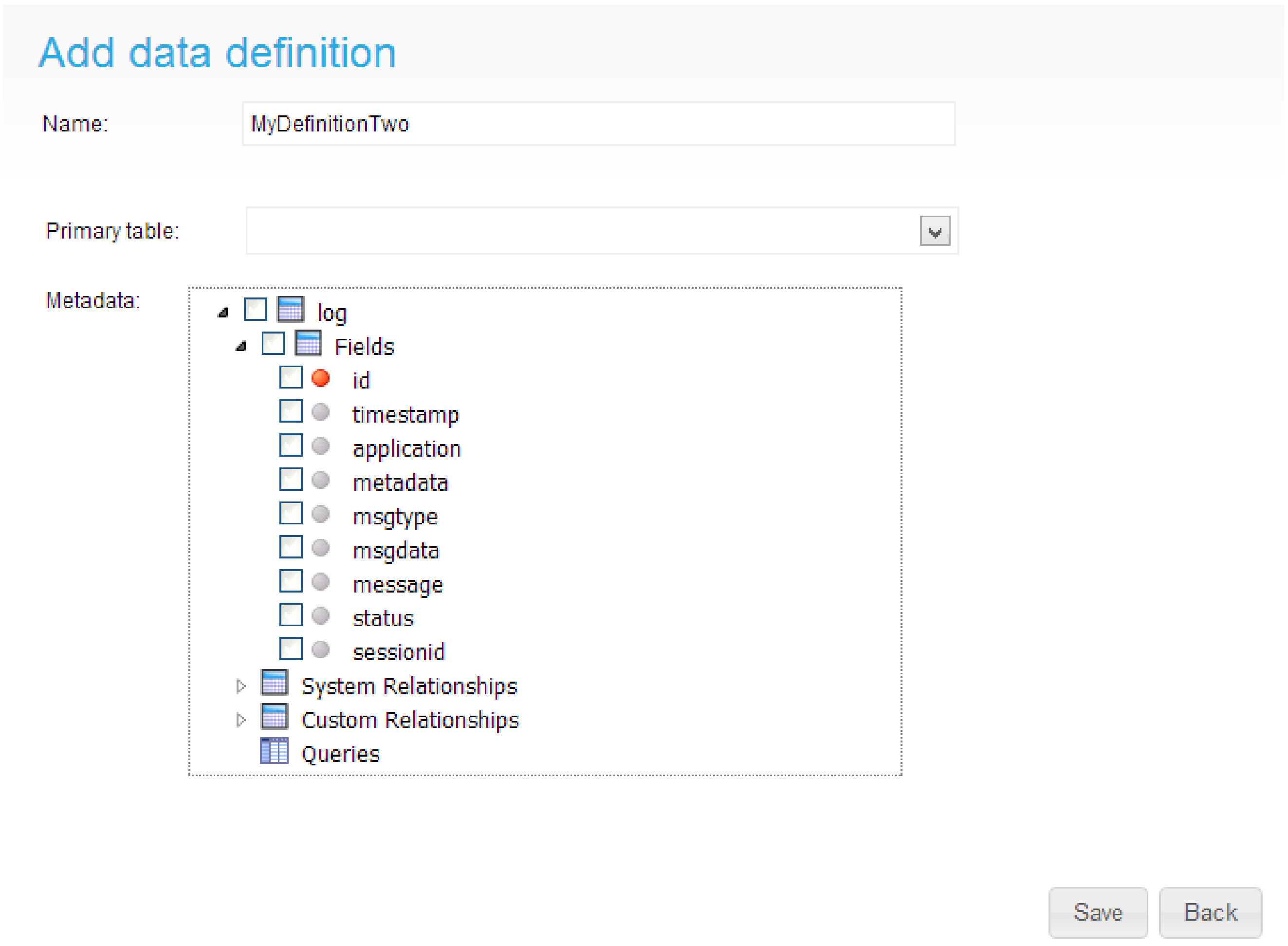
4.1.1 Data source

Selecting one of the data sources prompts you to select a schema (the default option is the one chosen during the data source creation process).


Add data definition

Name:	MyDefinitionTwo
Datasource:	Connectie Jan
Schema:	xblog

Click Next to bring up a data selection box.


Add data definition

Name: MyDefinitionTwo

Primary table:

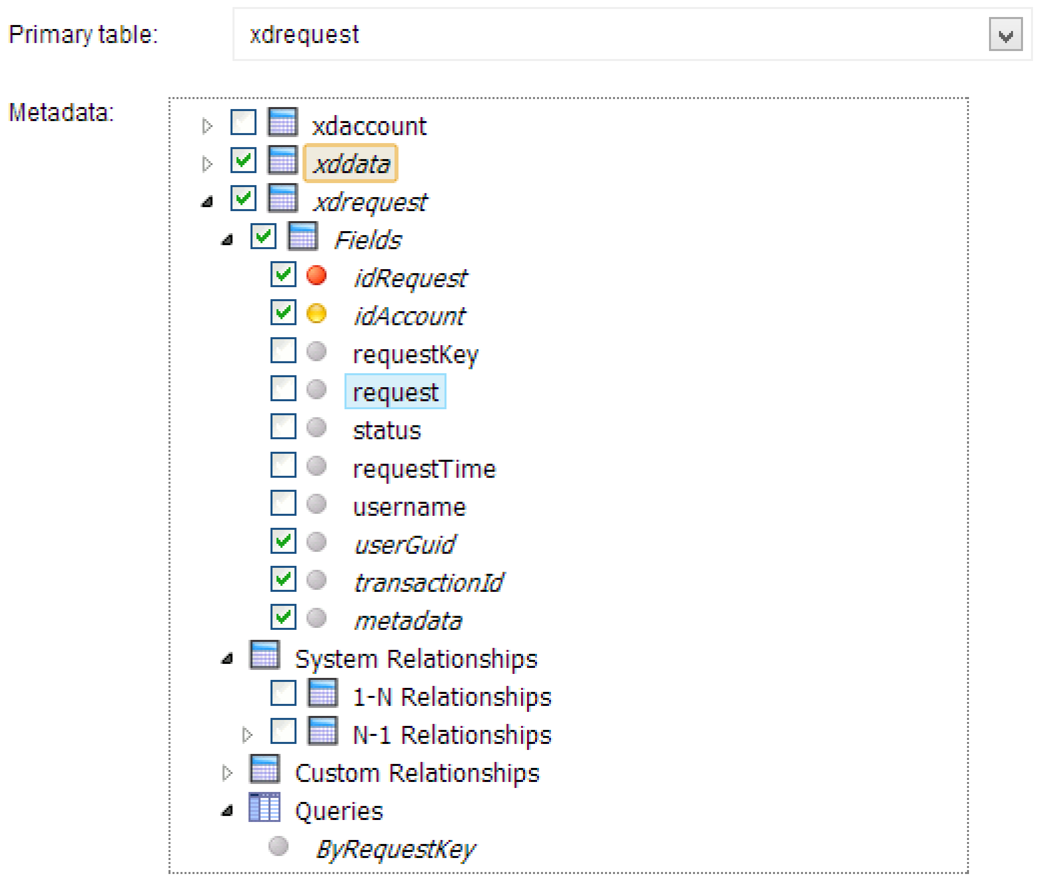
Metadata:

- log
  - Fields
    - id
    - timestamp
    - application
    - metadata
    - msgtype
    - msgdata
    - message
    - status
    - sessionid
  - System Relationships
  - Custom Relationships
  - Queries

In this box, you can select which fields and relationships you want to attach to the data definition. You can select and deselect a field/relationship/table by clicking on the box next to its name. It is important to note that, while you can select fields without selecting the parent element Fields or the parent table, that this doesn't work. If you want to attach an element to a data definition, you need to select all the parent elements as well.

Also note that fields which function as a primary key have a red dot in front of them, and fields which function as a foreign key have a yellow dot in front of them.

When you have selected the data you want to attach to the data definition, you need to choose the primary table you want to use.



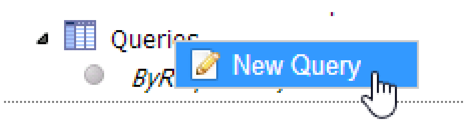
Primary table: xdrequest

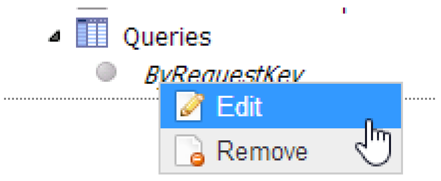
Metadata:

- ☐ xdaccount
- ✓ *xddata*
- ✓ *xdrequest*
  - ✓ *Fields*
    - ✓ *idRequest*
    - ✓ *idAccount*
    - ☐ requestKey
    - ☐ request
    - ☐ status
    - ☐ requestTime
    - ☐ username
    - ✓ *userGuid*
    - ✓ *transactionId*
    - ✓ *metadata*
  - System Relationships
    - ☐ 1-N Relationships
    - ☐ N-1 Relationships
  - Custom Relationships
  - Queries
    - *ByRequestKey*

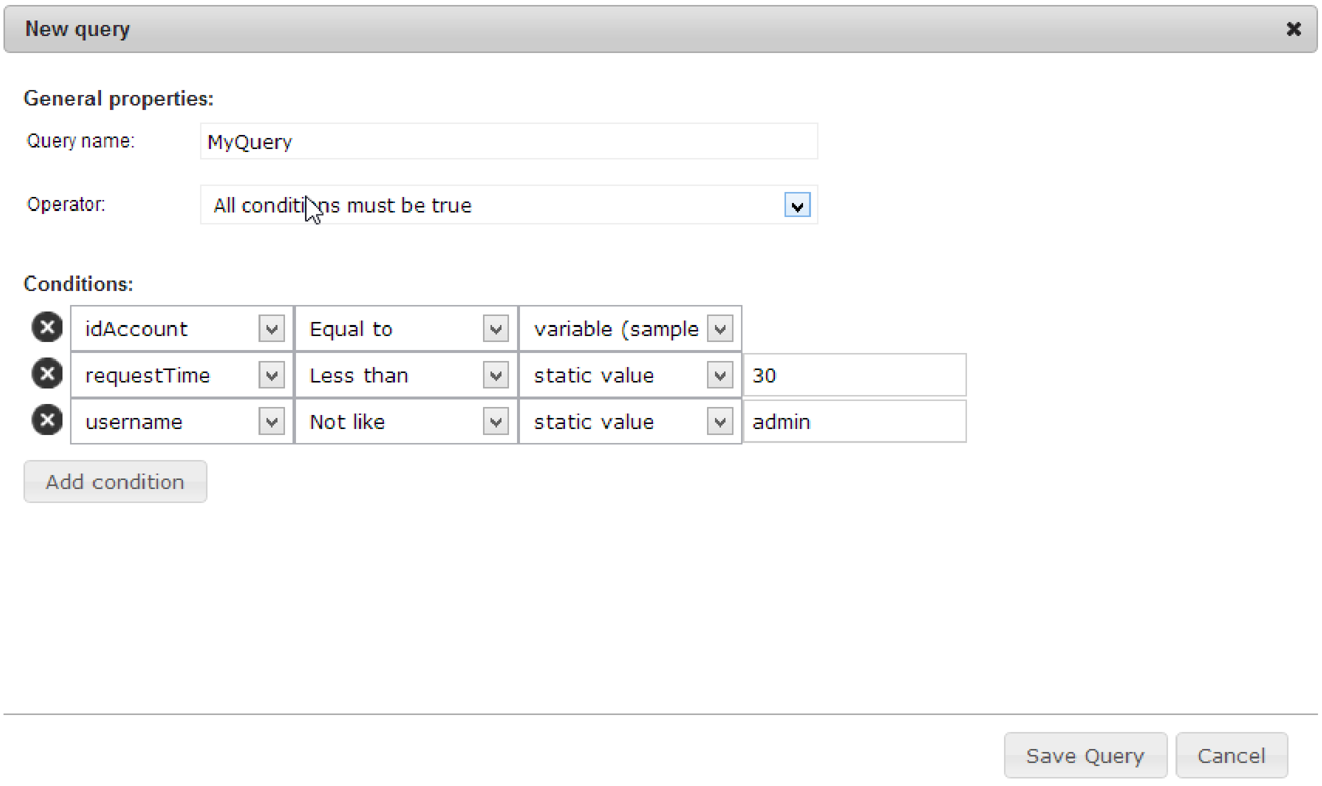
In this example, both the table xdrequest and xddata are selected. By selecting xdrequest in the drop-down list Primary table, only the elements from xdrequest will be usable. Thus, it is important to note that only one table's elements can be used.

You can also create queries to filter the data. To do this, right-click Queries. If there are already queries present, right-click an existing query to edit it.





A window will pop up with a few options:


New query

General properties:

Query name:	MyQuery
Operator:	All conditions must be true

Conditions:

idAccount	Equal to	variable (sample	
requestTime	Less than	static value	30
username	Not like	static value	admin

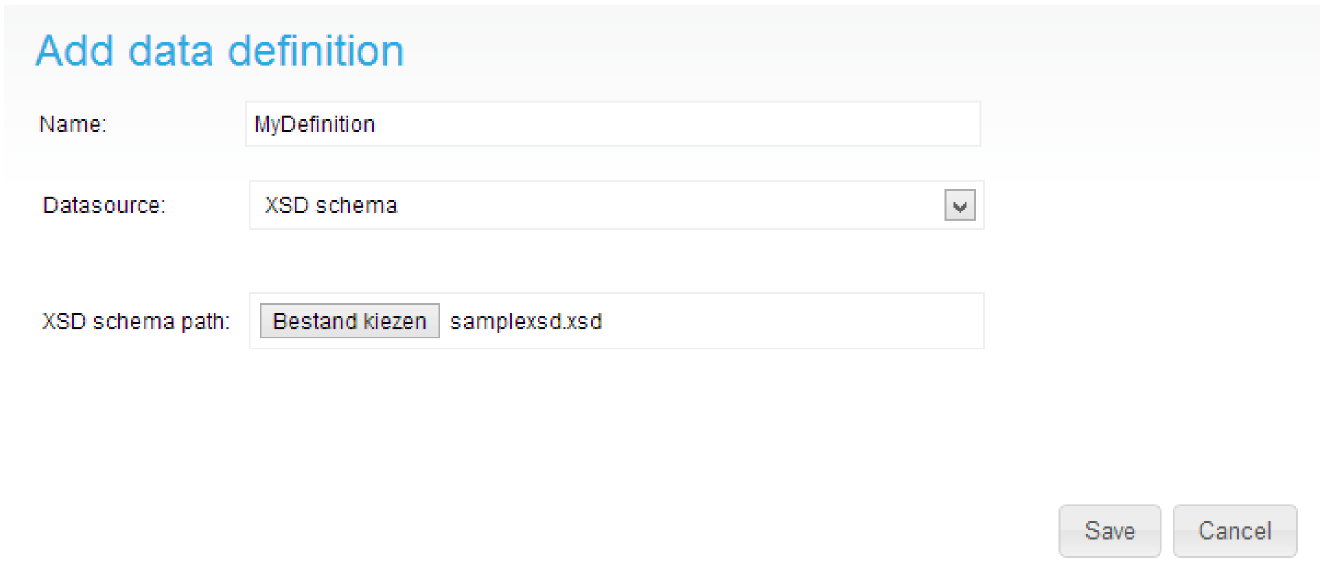
- Query name: the name of the query.
- Operator: selecting All conditions must be true will only attach data for which all conditions are true. Selecting Any one condition must be true means that data gets attached as soon as one of the conditions is met.

You can add conditions with the Add condition button. Each condition checks a field (left drop-down list) versus a static or variable value (right-drop down list) and requires an operator (middle drop-down list). In the above example, only data will be attached where requestTime is less than 30, where username is not like admin and where idAccount is equal to a variable value. This variable value can be defined when adding a sample, so that you can re-use the query but use different values each time.

Click Save Query to save the query.

4.1.2 XSD schema

Selecting XSD schema prompts you to browse for an XSD file.

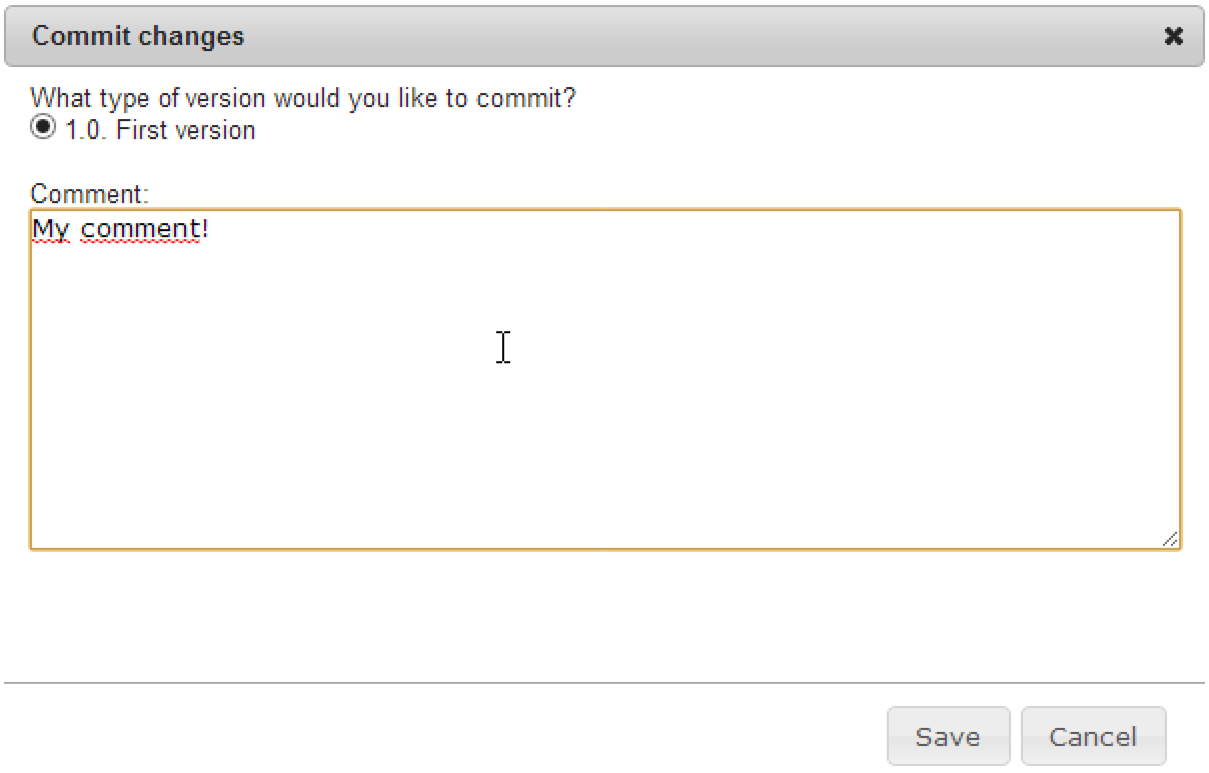

Add data definition

Name: MyDefinition

Datasource: XSD schema

XSD schema path: Bestand kiezen samplexsd.xsd

Click Save to save the data definition. A window will pop up, asking you which version you would like to commit. Since you are creating a new data definition, you can only select version 1.0. You can add a comment as well.


Commit changes

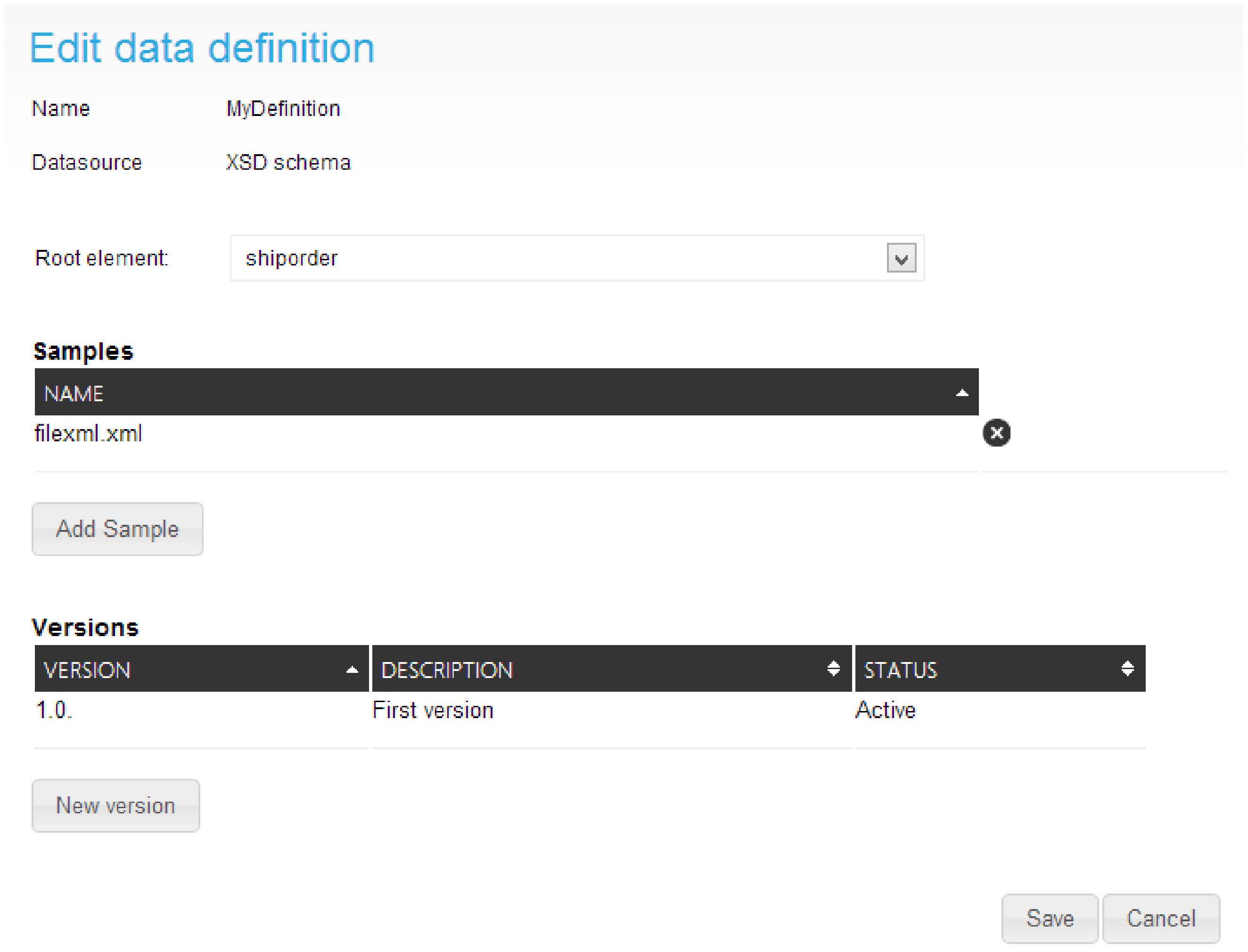
What type of version would you like to commit?
● 1.0. First version

Comment:
My comment!

Click Save to continue.

4.2 Editing a data definition

Editing a data definition is done by clicking on one of the definitions in the list. The following screen shows:


Edit data definition

Name	MyDefinition
Datasource	XSD schema
Root element:	shiporder

Samples

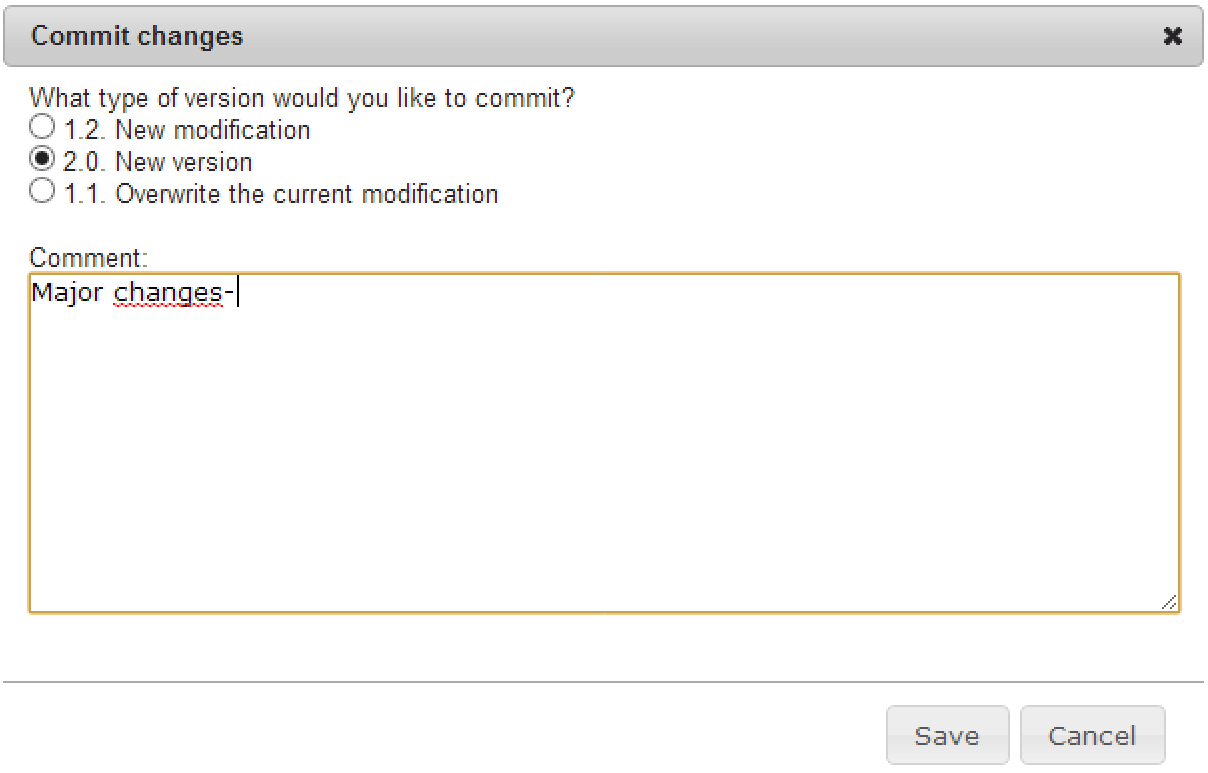
NAME
filexml.xml

Versions

VERSION	DESCRIPTION	STATUS
1.0.	First version	Active

4.2.1 Versions

You can add a new version of a data definition by clicking New version. Once saved, you are asked to supply a version number and a comment:



Commit changes

What type of version would you like to commit?

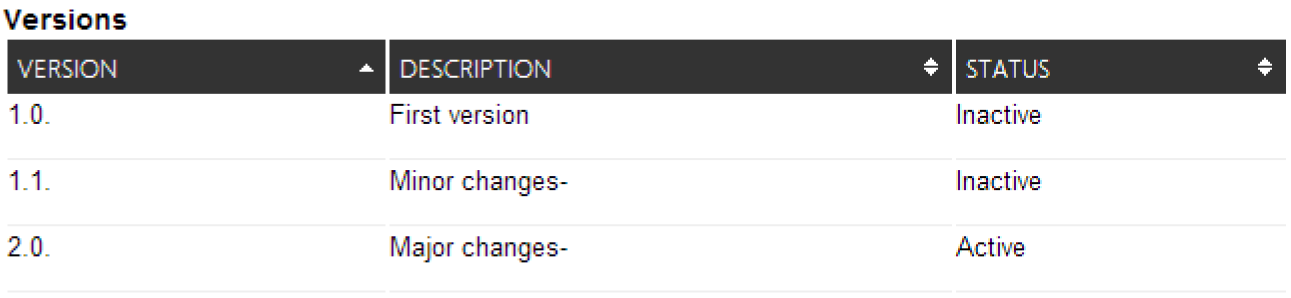
( ) 1.2. New modification
[✓ 2.0. New version]
( ) 1.1. Overwrite the current modification

Comment:

Major changes-

Save | Cancel

A new version number usually denotes a major change, while a new modification denotes a minor change. You can also overwrite the current version. These versions are kept in the database and you can view their properties by clicking on them in the list.



Versions

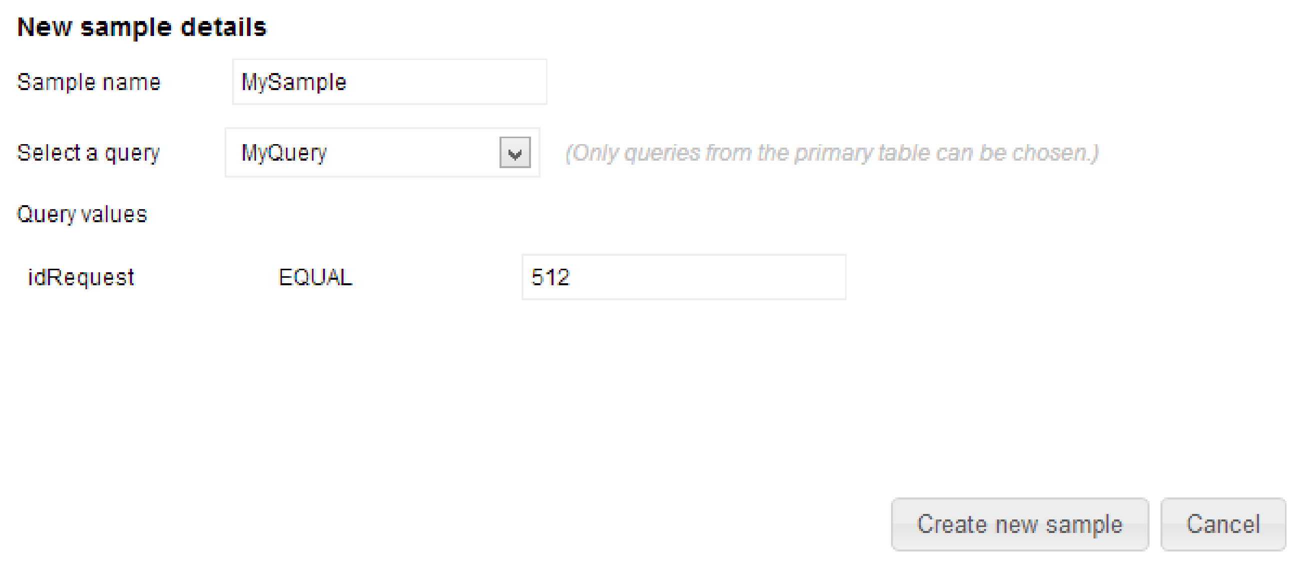
VERSION	DESCRIPTION	STATUS
1.0.	First version	Inactive
1.1.	Minor changes-	Inactive
2.0.	Major changes-	Active

4.2.2 Samples

You can add a sample by clicking Add Sample.

If this is a data definition from an XSD schema, you are asked for an XML file. This XML should be valid against the XSD schema.

If this is a data definition from a data source, you are asked for a sample name, a query and eventually a few values, depending on how many conditions in the query were defined to be dependant on variable values:



New sample details

Sample name	MySample	
Select a query	MyQuery	*(Only queries from the primary table can be chosen.)*
Query values		
idRequest	EQUAL	512

Click Create new sample to save the sample.

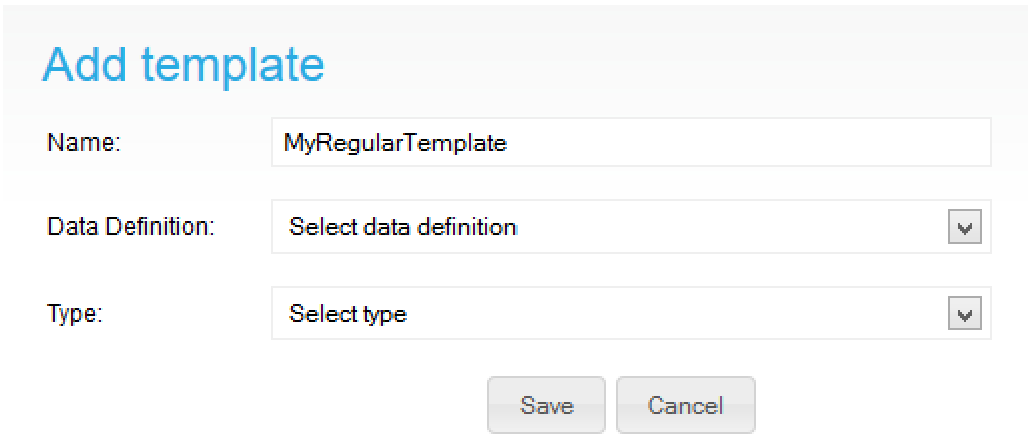
5 Templates

Templates are blueprints of your documents. They contain static content, placeholders for variable data and some intelligence around conditional content and formatting. Create new templates here, and design them with the XperiDo Template Design Add-In for Microsoft Word. Click Templates in the document flow controls to bring up a list of all the current data definitions.

There are two kinds of templates: regular and multipart. Regular templates define an entire document, whereas multipart templates consist of mulltiple header, footer, body and insert templates which can be placed conditionally.

5.1 Regular templates

Click New template to create a new regular template.


Add template

Name:	MyRegularTemplate
Data Definition:	Select data definition
Type:	Select type

In this screen, you are asked for a few properties:

- Name: the name of the template.
- Data Definition: the data definition on which this template is based. You can also choose static, which means that no variable data will be present in the template.
- Type: the type of template: header, footer, body or insert.

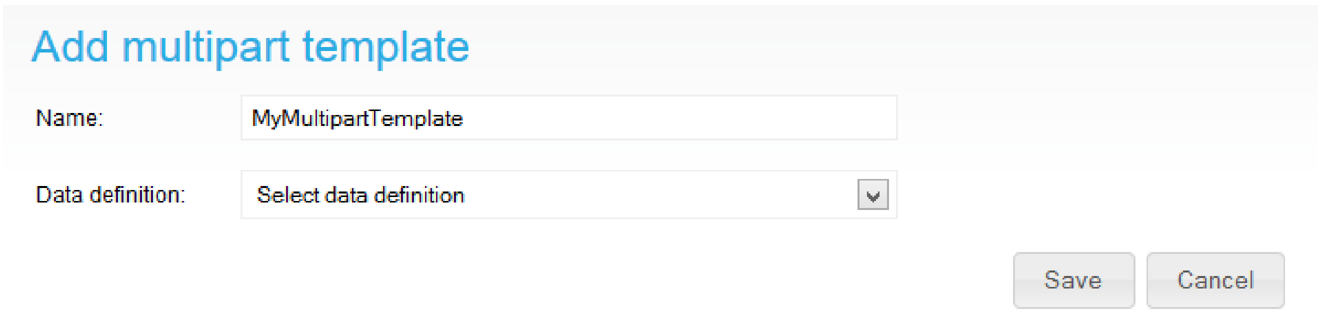
Header, footer and insert templates are meant for use in multipart templates, whereas body templates can also be stand-alone.

Click Save to save the template.

To preview your template (after you've designed it in Word), click it in the list of templates.

5.2 Multipart templates

Click New multipart template to create a new multipart template.


Add multipart template

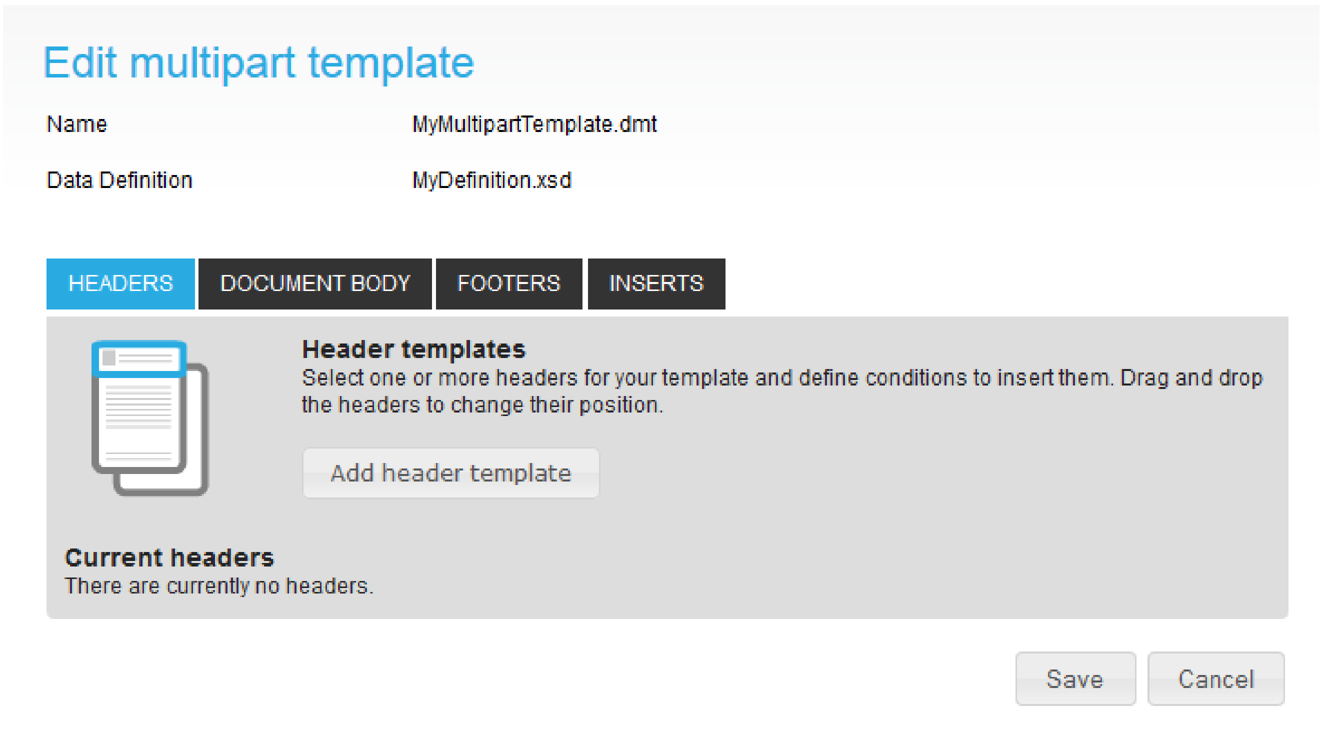
Name: MyMultipartTemplate

Data definition: Select data definition

In this screen, you are asked for a few properties:

- Name: the name of the template.
- Data Definition: the data definition on which this template is based.

Click Save to save the template. This will take you to the Edit multipart template screen, which can also be accessed by clicking a multipart template in the list of templates.

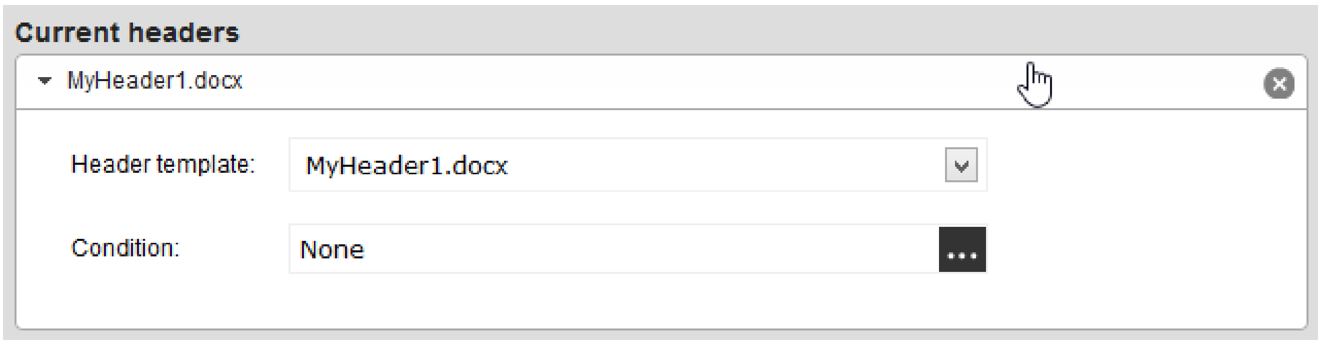

Edit multipart template

Name	MyMultipartTemplate.dmt
Data Definition	MyDefinition.xsd

Header templates
Select one or more headers for your template and define conditions to insert them. Drag and drop the headers to change their position.

Current headers
There are currently no headers.

In here, you decide which headers, bodies, footers and insert to use in your multipart template. Click on a section and click Add header/body/footer/insert template to add regular templates. You can only add templates that are based on the same data definition.

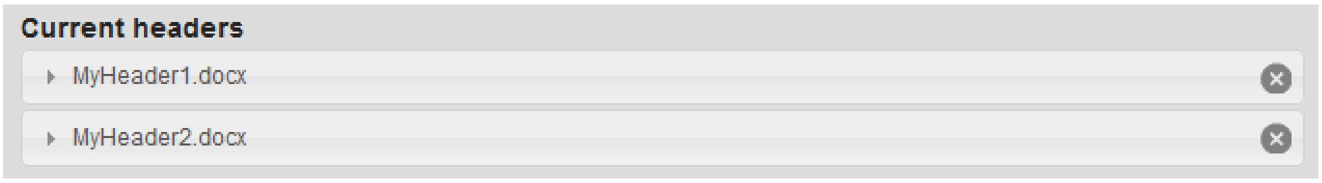

Current headers

MyHeader1.docx

Header template: MyHeader1.docx

Condition: None

You can add multiple templates of the same type:


Current headers

MyHeader1.docx

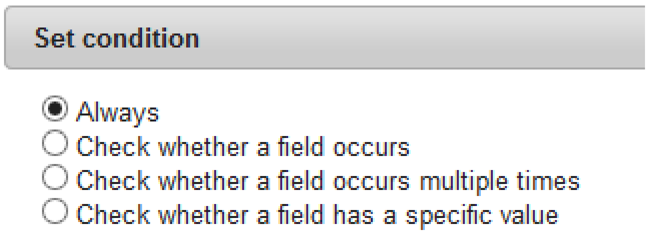
MyHeader2.docx

The order in which templates are placed determines which template will be used. In this case, MyHeader1 will always be used as the header, since it is placed above MyHeader2. You can change the order of the templates by clicking and dragging them in the position you want them to be.

5.2.1 Conditional templates

Each template you add can have a condition associated with it, which defines when this template will be used. This makes it possible to have the same multipart template output different headers for different input data.

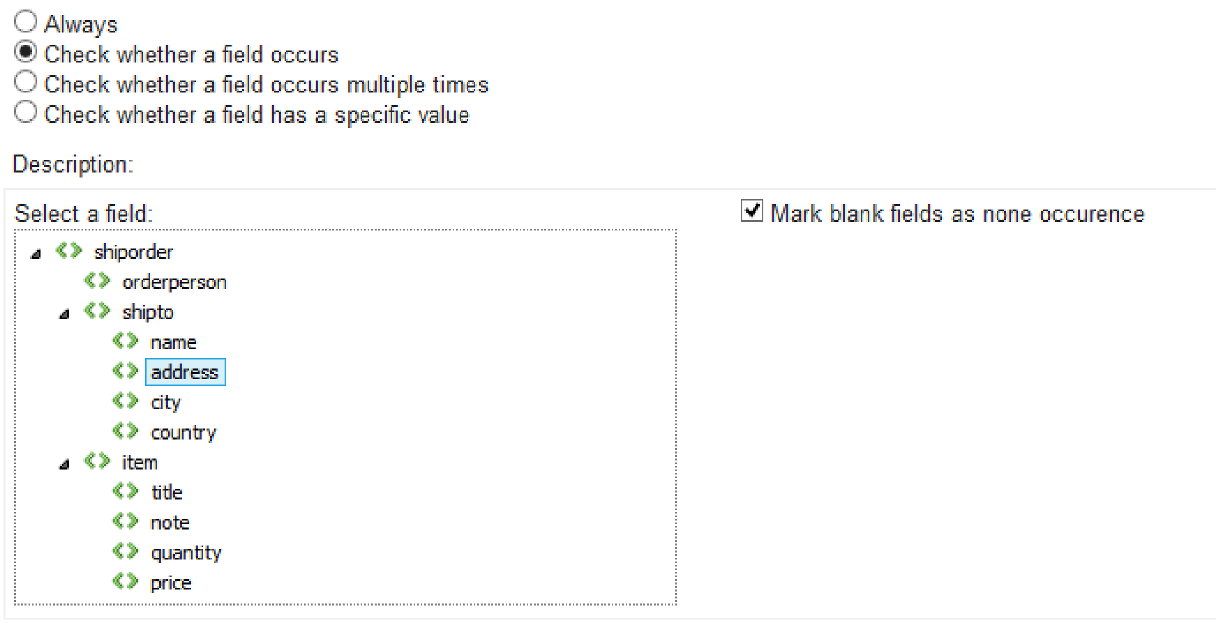
Click the ... to select a condition. The following window pops up:


Set condition

[✓ Always]
Check whether a field occurs
Check whether a field occurs multiple times
Check whether a field has a specific value

In here, you set the condition upon which the visibility of this template is based:

- Always: the template is always used.
- Check whether a field occurs: the template will only be used if a field in the data definition occurs.


- ○ Always
- ● Check whether a field occurs
- ○ Check whether a field occurs multiple times
- ○ Check whether a field has a specific value

Description:

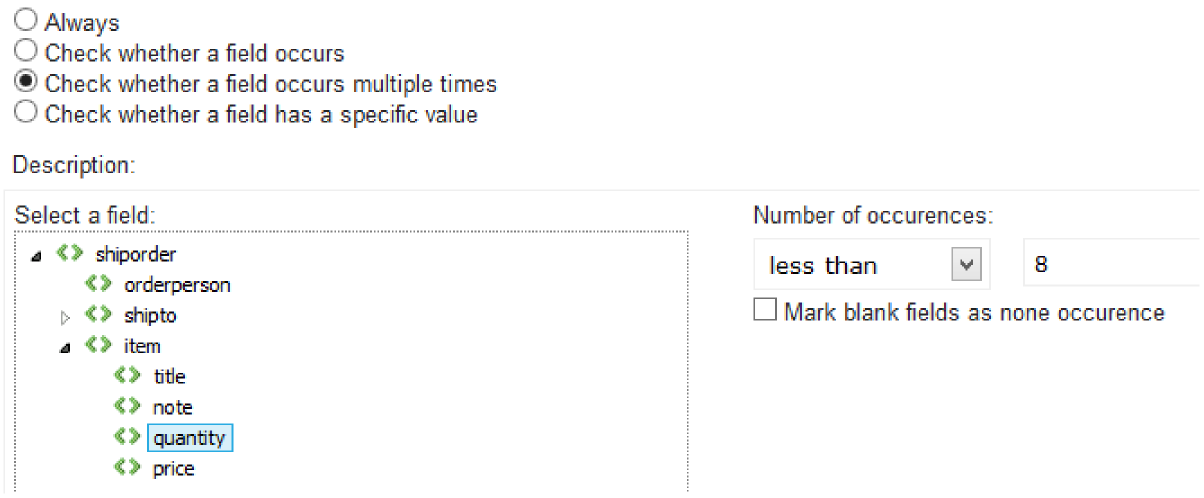
Select a field:

- shiporder
  - orderperson
  - shipto
    - name
    - address
    - city
    - country
  - item
    - title
    - note
    - quantity
    - price

☑ Mark blank fields as none occurence

On the left, you select the field you wish to check. The right checkbox that says Mark blank fields as none occurence treats empty fields as not occuring.

- Check whether a field occurs multiple times: the template will only be used if a field in the data definition occurs multiple times.


- ○ Always
- ○ Check whether a field occurs
- ● Check whether a field occurs multiple times
- ○ Check whether a field has a specific value

Description:

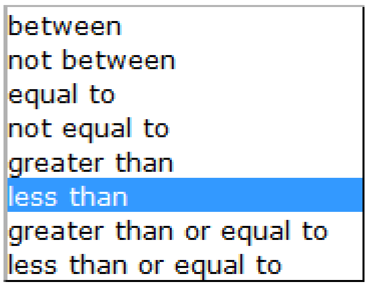
Select a field:

- shiporder
  - orderperson
  - shipto
  - item
    - title
    - note
    - quantity
    - price

Number of occurences: less than 8

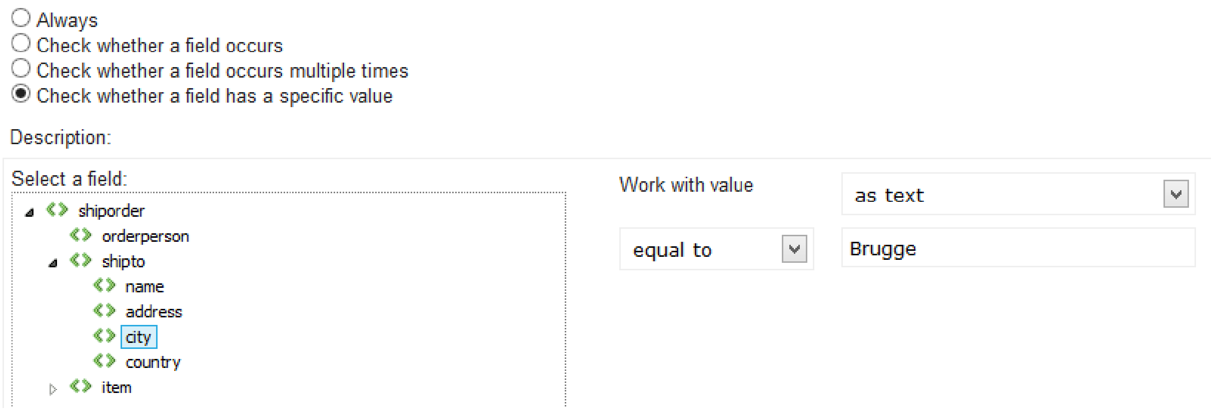
☐ Mark blank fields as none occurence

On the left, you select the field you wish to check. The right checkbox that says Mark blank fields as none occurence treats empty fields as not occuring. Above this checkbox you can select the operator, which is one of the following:


between
not between
equal to
not equal to
greater than
less than
greater than or equal to
less than or equal to

Next to the operator, you can input the number to check against. In the example above, the header will only be used if the value of the field quantity is less than 8.

- Check whether a field has a specific value: the template will only be used if a field in the dataset has a specific value.


Always [✓ Check whether a field has a specific value] — Check whether a field occurs — Check whether a field occurs multiple times

Description:

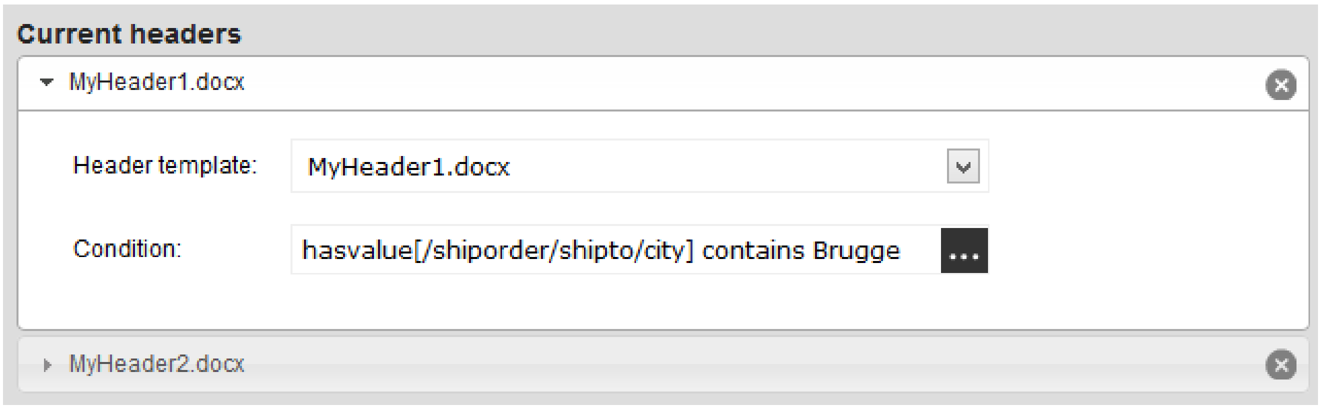
Select a field:
- shiporder
  - orderperson
  - shipto
    - name
    - address
    - city
    - country
  - item

Work with value: as text

equal to: Brugge

On the left, you select the field you wish to check. On the right, you choose whether to work with this value as text, as number or as a date. The drop-down list and the text input field below change accordingly. In the example above, the header will only be used if the value of the field city is equal to Brugge.

Assume we added 2 headers, with the first header having a condition as follows:


Current headers

MyHeader1.docx

Header template: MyHeader1.docx

Condition: hasvalue[/shiporder/shipto/city] contains Brugge

MyHeader2.docx

In this case, MyHeader2 will always be shown (since it has no condition), unless the data definition's city field contains Brugge, in which case MyHeader1 is shown.

Select one or more header/body/footer/insert templates and click Save to save the template.

5.3 Assets

Assets are images that you can use in your templates. While it is possible to insert images in Word from anywhere, you can store images in the repository and retrieve them through the asset manager with XperiDo for Microsoft Word. Click Assets in the document flow controls to bring up a list of all the images that are currently in the repository.

Click New Asset to bring up a screen where you can browse for an image. Click Add to add it to the repository.

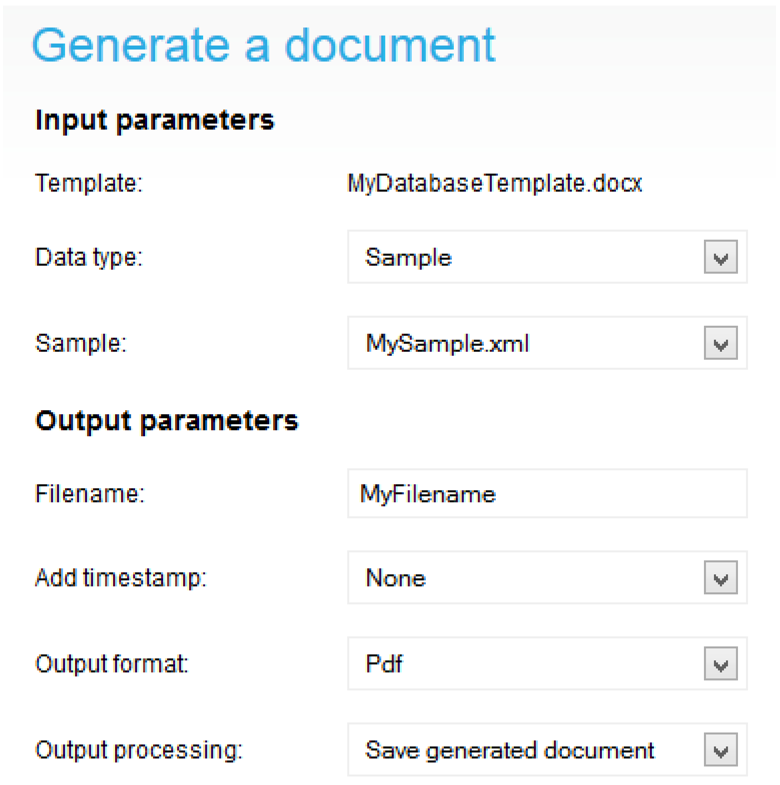
6 Generate

With XperiDo Nucleus, you can generate documents. Either you use the Nucleus interface to generate a document, where the input data comes from a sample, query or xml file, or you use the associated webservices for the creation of multiple documents at once, with input data added as you see fit.

Click Generate in the document flow controls to bring up a list of all the current templates.

6.1 With XperiDo Nucleus

Click a template to go to the following screen:



Generate a document

Input parameters

Template:	MyDatabaseTemplate.docx
Data type:	Sample
Sample:	MySample.xml

Output parameters

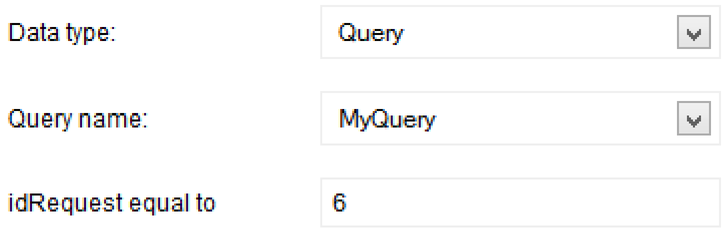
Filename:	MyFilename
Add timestamp:	None
Output format:	Pdf
Output processing:	Save generated document

6.1.1 Input parameters

Template shows the name of the selected template. A preview is shown to the right.

Data type defines which data will be used to fill in the placeholders in the template. The contents of this drop-down list depend on which type of data definition this is (either a data source or an XSD schema). In either case, you can select Sample, which lets you choose a sample from the available samples in the drop-down list next to Sample.

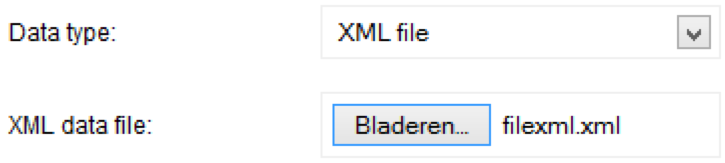
- If the data definition type is a data source (database connection), then you can select Query. Then, next to Query name, you can choose Select all records, which selects all the records as input data. You can also choose out of the existing queries, and then assign a value to the conditions where you specified the value as variable:



Data type:	Query
Query name:	MyQuery
idRequest equal to	6

In this example, we defined a query called MyQuery during the data definition process, and we set the value of one of the conditions to be variable. This condition checks whether the field idRequest is equal to a number, in our case 6. By setting query's condition's values as variable, you can re-use the same query with a different value every time you generate a document.

- If the data definition type is an XSD schema, then you can select an XML file to use:



Data type:	XML file
XML data file:	Bladeren... filexml.xml

Click the browse button and select a file to use its data for this document.

6.1.2 *Output parameters*

Filename is the name of the generated document.

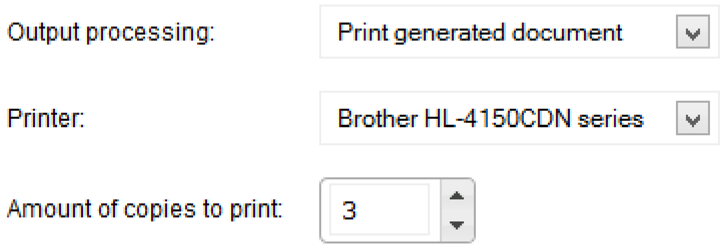
Add timestamp asks you whether you want to add a timestamp to the name of the document.

- YYYYMMDD: the date is added in the following format: 19th of October 2016 becomes 20131019.
- YYYYMMDD_HHMMSS: the date and time are added in the following format: 19th of October 2013, 15:26:05 becomes 20131019_152605

Output format determines in which format the document will be output. The following formats are available: pdf, docx, doc, rtf, html, jpg, gif, tiff, odt, epub, xps.

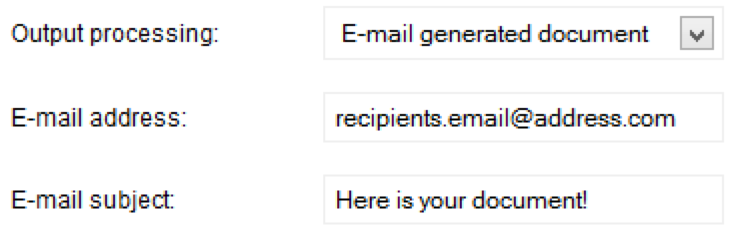
Output processing determines what should be done with the document. It can be:

- Saved: the document will be stored in a temporary location, after which you can download it to your computer.
- Printed: the document will be printed a chosen number of times on a printer that you can choose from the list of available printers in your network.



Output processing:	Print generated document
Printer:	Brother HL-4150CDN series
Amount of copies to print:	3

- E-mailed: the document will be emailed to the chosen e-mail address.



Output processing:	E-mail generated document
E-mail address:	recipients.email@address.com
E-mail subject:	Here is your document!

Click the Generate button to generate your document.

6.2 Through the use of webservices

You can generate documents through the use of webservices. How to do this is explained in the section Using webservices.

7 Using webservices

XperiDo Nucleus consists of two components: a webservices interface and a management console . The management console is a website that offers a simple user interface to easily perform basic CRUD operations like creating and deleting data sets, samples, templates, markers etc. The webservices interface form a bridge between your company's own application and all the XperiDo data created in the management console.

Currently XperiDo Nucleus supports REST webservices. This part of the manual explains which webservices are available, which input they consume, which parameters they need and which output they produce. The fundamental knowledge of how to invoke a webservice is assumed to be known by developers using these webservices. However, a Java and a .NET API library is available to invoke the webservices in an object oriented way, without having to worry about the details of setting up a http(s) connection first.

7.1 Webservice URL

A webservice's full url is formed as in the following example. Parts between <> are meant to be filled in by specific values, which can differ between each service.

```
http(s)://<servername>:<port>/<path_to_webservices>/<webservice_name>/{<resource_id>}<query_string>
```

- <servername>: The name of the server (sometimes called host) where the webservices can be invoked. This name is the same for all webservices.
- <port>: The portnumber on that server that will accept incoming request to invoke the webservices. This port is the same for all webservices. Depending on the port configuration, the port will accept secured (https) or unsecured (http) connections, but never both. It is possibly to configure two ports, one which can handle secured connections and one which can handle unsecured connections.
- <path_to_webservices>: The path on the server where the webservices can be invoked. This path is the same for all webservices. By default, this part of the url is "xbi/rest/xperido/" (note that xperido is written in lowercase).
- <webservice_name>: The name of the webservice itself. Webservice names are camelCased and the casing must match in the url.
- <resource_id>: Resource ids are only required by some webservices. Other webservices do not have this part in their full url. They denote the unique id of an item, and can thus be different for each service invocation. They are usually obtained by first calling another webservice which returns one or more resource ids in its response. In theory, a webservice can have any amount of resource ids in its url, but the current set of webservices never have more than one resource id.
- <query_string>: Some webservices can be given query parameters to finetune their behaviour or to provide necessary information. A query string always starts with "?" and can then contain any amount of "<parameter_name>=<paramater_value>" pairs. Each pair is separated by a "&" sign. An example query string is given below. If a service requires parameters, the amount of parameters and their name must match exactly. The parameter names are camelCased and their casing must match in the url.

```
?template=MyTemplate.docx&templateType=DocTemplate& ...
```

The length of an URL is limited. Because of this, if a webservice requires a potentially large amount of data, this data cannot be transferred using only query parameter(value)s. For example, the webservice with name "createDocumentWithData" requires (a potentially large amount of) xml data to be send to the server. To avoid this problem, the data must be sent as part of the request body instead of the request header where query parameters reside. In this manual, whenever data must be sent in the request body, we refer to that data as "input" of the webservice, and we say that the webservice "consumes" input. For each service that requires input, the type of expected input is specified.

All services produce an output response. For each service, the type of output is specified and an example of this output is given. Some webservices return a stream of bytes as response, but most webservices return either a JSON or XML formatted responses. Developers are free to choose which output format best suits their needs. However, caution is required when choosing the JSON output format. Internally, the service always produces XML. This means that when choosing the JSON output format a conversion has to take place. The amount of time that this conversion requires is negligible even for large responses. The problem with the conversion is that information about "arrays" changes depending on the amount of items in the array. Take for example the following XML snippet:

```
<Templates>
  <Template>

    ...data1...

  </Template>
  <Template>

    ...data2...

  </Template>

</Templates>
```

```
When converted to JSON, this would become:
```

```
{"Templates":{"Template":[{...data1...},{...data2...}]}}
```

```
However, if the amount of <Template> elements in the xml changes, the JSON changes
internally:
```

```
<Templates>

  <Template>

    ...data1...

  </Template>
```

```
</Templates>

Becomes a JSON string without an array:

{"Templates":{"Template":{...data1...}}}

And if no <Template> elements are present:

<Templates>
</Templates>

The JSON string will not contain an inner object:

{"Templates":""}

This means that parsing JSON output must be done carefully, as assumptions about
the presence or the type of elements can be wrong.
```

7.2 Available webservices

The following webservices can be used with XperiDo Nucleus:

7.2.1 listDatasets

The listDatasets service returns all the data sets that are defined in the project.

- **Resource IDs**: Does not require any resource IDs.
- **Query Parameters**: Does not require any query parameters.
- **Input**: Does not consume any input.
- **Output:** Produces a response string of type "*application/json*".

Example output:

```
{
 "ListDataSetsResponse":
 {
  "DataSets":
  [
   {
    "Description": "Main invoice data set.",
    "FullDataSetName": "Projectname/Foldername/Invoice.xsd",
    "DataSetName": "Invoice.xsd"
   }
   {
    "FullDataSetName": "Projectname/Foldername/Static.xsd",
    "DataSetName": "Static.xsd"
   }
  ]
 }
}
```

In this example, there are 2 datasets: Invoice and Static. The second data set has no description set, so the output response will not contain a description element for it.

7.2.2 listTemplates

The listtemplates service returns some basic information about templates in the project. An XML string must be passed as input. This input describes a query expression (in XML) to filter templates on marker values. If the given input is the empty string, all templates in the project will be returned and no expression is evaluated

- **Resource IDs**: Does not require any resource IDs.
- **Query Parameters**: Does not require any query parameters.
- **Input**: Consumes a string of type *"application/XML"*.

  This string must be added to the http request BODY, not the header.

Example input:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Expression>
  <_or>
    <condition>
      <name>SomeMarkerName</name>
      <value>SomeMarkerValueName</value>
      <operation>eq</operation>
    </condition>
    <condition>
      <name>AnotherMarkerName</name>
      <value>AnotherMarkerValueName</value>
      <operation>eq</operation>
    </condition>
  </_or>
</Expression>
```

- **Output:** Produces a response string of type *"application/json"*. This string contains different information depending on whether an input query expression was given or not (empty string).

Example output with empty string input (no query expression will be evaluated, and thus all the templates will be returned):

```
{
 "ListTemplatesResponse":
 {
  "Templates":
  [
   {
    "FullQueryName": "/Projectname/Datasets/Invoice.xsd",
    "TemplateType": "DocTemplate",
    "TemplateName": "Invoice.docx",
    "FullTemplateName": "/Projectname/Templates/Invoice.docx"
   }
  ]
 }
}
```

In this example, there is only 1 template, named Invoice.docx, of type DocTemplate and based on the Invoice.xsd dataset.

Example output with a non-empty valid query expression (only templates matching the query expression will be returned):

```
{
 "ListTemplatesResponse":
 {
  "Templates":
  [
   {
    "TemplateName": "Invoice.docx",
    "FullTemplateName": "/Projectname/Templates/Invoice.docx"
   }
  ]
 }
}
```

When the query expression is not empty, only the TemplateName and FullTemplateName are returned.

7.2.3 createDocument

The createDocument service creates a document only for templates whose data set is based on a data source (database connection) rather than an XSD schema. (For XSD schemas, the equivalent createdocumentwithdata service can be used.)

- **Resource IDs**: Does not require any resource IDs.
- **Query Parameters**: Requires a total of 11 query parameters.

1) template: the name of the template with or without the extension (which is either "*.docx*" or "*.dmt*").
2) templatetype: the ITEM type of the template (either "*DocTemplate*" or "*DocMergeTemplate*"), not to be confused with the TEMPLATE type of the template (such as body, header, footer, multipart...). Can be empty ("") if the file extension of the name is given.
3) query: the name of the query.
4) key: a string that holds all the variables in the query
5) saveLocation: the location on the server where the created document will be stored.
6) saveName: the name of the generated document.
7) saveFormat: the output format of the generated document. ("Bmp","Doc","Docm","Docx","Dot","Dotm","Dotx","Emf","Epub","FlatOpc","Html","Jpeg","Mhtml","Odt","Ott","Pdf","PdfA1b","Png","Rtf","Text","Unknown","WordMl","XamlFlow" or "Xps")
8) mailTo: the e-mailaddress to mail the generated document.
9) mailSubject: the subject line of the e-mail.
10) printPrinter: the name of the network printer to print on.
11) printCopies: the amount of copies to print.

- **Input**: Does not consume any input.
- **Output:** Produces a response string of type "*application/json*".

Example output:

```
{
 "CreateDocumentResponse":
 {
  "document":
  {
   "path": "C:\\path\\to\\file\\MyGeneratedDocument.pdf",
   "mimeType": "application/pdf"
  },
   "requestId": "39e38a94-cd02-4ae3-afa0-dd1f3ee7c3cc",
  "finished": true
 }
}
```

In this example, the finished parameter refers to the status of the process. The requestId parameter refers to the unique transaction number created by the system and can be used to retrieve the document or for a follow-up when the finished parameter is false (see the retrieveDocument services below).

7.2.4 createDocumentWithData

The createDocumentWithData service creates a document for any template (regardless of the type of data set). To provide data, an XML (string) must be supplied.
This XML (string) must be in accordance to the structure of the data set of the template.

- **Resource IDs**: Does not require any resource IDs.
- **Query Parameters**: Requires a total of 9 query parameters.

1) template: the name of the template with or without the extension (which is either "*.docx*" or "*.dmt*").
2) templatetype: the ITEM type of the template (either "*DocTemplate*" or "*DocMergeTemplate*"), not to be confused with the TEMPLATE type of the template (such as body, header, footer, multipart…). Can be empty ("") if the file extension of the name is given.
3) saveLocation: the location on the server where the created document will be stored.
4) saveName: the name of the generated document.

5) saveFormat: the output format of the generated document. ("Bmp","Doc","Docm","Docx","Dot","Dotm","Dotx","Emf","Epub","FlatOpc","Html","Jpeg","Mhtml","Odt","Ott","Pdf","PdfA1b","Png","Rtf","Text","Unknown","WordMl","XamlFlow" or "Xps")

6) mailTo: the e-mailaddress to mail the generated document.

7) mailSubject: the subject line of the e-mail.

8) printPrinter: the name of the network printer to print on.

9) printCopies: the amount of copies to print.

- **Input**: Consumes a string of type *"application/XML"*.

This string must be added to the http request BODY, not the header.

- **Output:** Produces a response string of type *"application/json"*.

Example output:

```
{
 "CreateDocumentResponse":
 {
  "document":
  {
   "path": "C:\\path\\to\\file\\MyGeneratedDocument.pdf",
   "mimeType": "application/pdf"
  },
   "requestId": "39e38a94-cd02-4ae3-afa0-dd1f3ee7c3cc",
  "finished": true
 }
}
```

The finished parameter refers to the status of the process. The requestId parameter refers to the unique transaction number created by the system and can be used to retrieve the document or for a follow-up when the finished parameter is false (see the retrieveDocument services below).

7.2.5 retrieveDocument/{request_ID}

The retrieveDocument service allows you to retrieve a document, as a stream, by supplying the unique request ID.

- **Resource IDs**: requires the unique request ID as returned by the createDocument or createDocumentWithData services.

Request Id Example: *"49ac0bbc-832f-49ef-8ade-7121554ee719"*.

- **Query Parameters**: Does not require any query parameters.

- **Input**: Does not consume any input.
- **Output:** Produces a response string of type *"application/joctet-stream"*. This represents the requested document as a base64 encoded stream.

The output is usually human unreadable. Saving the output in a file, and giving that file the right extension (for example, pdf) should in effect reconstruct the file.

7.2.6 retrieveDocumentAsMessage/{request_ID}

The retrieveDocumentAsMessage service allows you to retrieve a document along with metadata (such as it's MIME-type) by supplying the unique request ID.

- **Resource IDs**: requires the unique request ID as returned by the createDocument or createDocumentWithData services.

Request Id Example: *"49ac0bbc-832f-49ef-8ade-7121554ee719"*.

- **Query Parameters**: Does not require any query parameters.
- **Input**: Does not consume any input.
- **Output:** Produces a response string of type *"application/json"*.

Example output:

```
{
  "CreateDocumentResponse":
  {
    "document":
    {
      "data":"JVBERi0xLjUNCjQg5 ... (usually a long string...) ... HJlZg0KNjMxDQolJUVPRg0K",
       "path":"C:\\temp\\xdbase\\MyLittleDocument_3-3-2014_17.12.36.584.pdf",
       "mimeType":"application/pdf"
    },
    "requestId":"212b46f4-656a-4d0d-8f7e-5fed9e9a3a57",
    "finished":false
  }
}
```

This output is the same the as for the createDocument and createDocumentWithData services, except now the document element also has an additional data element. The document - data element is a Base64 encoded string that holds the content of the created document file.

7.2.7 retrieveCreateDocumentWithDataStatus/{request_ID}

The retrieveCreateDocumentWithDataStatus service allows you to check on the status of a previous createDocumentWithData service invocation. This is useful if a previous invocation of the createDocumentWithData service returned a response with status "finished = false".

- **Resource IDs**: requires the unique request ID as returned by the createDocumentWithData service.

Request Id Example: *"49ac0bbc-832f-49ef-8ade-7121554ee719"*.

Note: This service will not work with the request id's returned by the createDocument service!

- **Query Parameters**: Does not require any query parameters.
- **Input**: Does not consume any input.
- **Output:** Produces a response string of type *"application/json"*.

  Example output:

```
{
 "RetrieveCreateDocumentWithDataStatusResponse":
 {
  "document":
  {
   "path": "C:\\path\\to\\file\\MyGeneratedDocument.pdf",
  },
  "finished": true
 }
}
```

This output is the similar to the output of the createDocument and createDocumentWithData services, except it has a different name and only the "document-path" and "finished" elements are set.

7.3 WADL

XperiDo Nucleus provides a WADL for more information about the semantics and hierarchy of the REST webservices. By default, this can be accessed by going to <xperidoserver>:<port>/rest/application.wadl. By default, <port> is 8080 (This is not necessarily the same port used to invoke the webservices!). The path, name and portnumber might be different depending on the configuration and installation of xperido on your server.

7.4 JAVA SDK

7.4.1 Getting started

A zip file containing three .jar files is available which contains all the necessary classes to invoke the XperiDo Nucleus webservices in a object-oriented way. To use it, unzip it and import the xperido-client.jar and the commons-codec-1.9.jar file (found in the lib folder) in your project. To ease the development process, you can configure your IDE to link the xperido-client.jar to its javadoc found in xperido-client-javadoc.jar

Next, instantiate the XperiDoNucleusServices class found in the com.invenso.services package of the xperido-client.jar. The constructor of this class requires some parameters:

_serviceBaseURL: This is the part of the webservice url that all webservices have in common. It is a string formed as such:

"http(s)://<servername>:<port>/<path_to_services>/"

- <servername>: The name of the server where the services can be invoked. This name is the same for all services. This name will be given to you by the xperido team.
- <port>: The portnumber on that server that will accept incoming request to invoke the rest. This port is the same for all services. Depending on the port configuration, the part will only accept secured (https) or unsecured (http) connections. This portnumber will be given to you by the xperido team
- <path_to_services>: The path on the server where the services can be invoked. This path is the same for all services. Unless stated otherwise, this part of the url is "xbi/rest/xperido/".

username: This parameter will be given to you by the xperido team.

password: This parameter will be given to you by the xperido team.

_ignoreHostnameVerification: When using a secured connection (https, SSL), it is possible the <servername> in the _serviceBaseURL parameter does not match the server's certificate hostname. If this parameter is set to false, in that case the connection will fail, because hostnames must match by default . If this parameter is set to true, hostnames in the certificate are ignored and the secured connection will work.

_additionalCertificatePath: It is possible the server uses self-signed certificates. When using a secured connection (https, SSL) in that case, the connection will fail because certificates must be signed by a CA by default. Setting this parameter to the path where a local copy of the server's certificate can be found (typically a .cert file), that certificate will also be considered trustworthy and the connection will work. If this is not necessary, this parameter can be null.

7.4.2 Invoking a webservice

The XperiDoNucleusServices class provides public methods for each web service. Developers can choose the type of response they receive from the web service by calling the appropiate methods. for example, the listtemplates webservice can be invoked by calling either of the following 6 methods:

1) public String getAllTemplatesAsJSON() - output: a JSON string.
2) public String getAllTemplatesAsXML() - output: an XML string.
3) public ListTemplatesResponse getAllTemplates() - output: a ListTemplatesResponse object.
4) public String getTemplatesByMarkerQueryAsJSON(Expression expression) - input: An Expression object, output: a JSON string.
5) public String getTemplatesByMarkerQueryAsXML(Expression expression) - input: An Expression object, output: an XML string.
6) public ListTemplatesResponse getTemplatesByMarkerQuery(Expression expression) - input: An Expression object, output: a ListTemplatesResponse object.

(*Note: the names of the methods in the XperiDoNucleusServices class are chosen to reflect what they do. They are not exactly equal to the names of the webservices themself, but it is easy to identify which methods(s) invoke which webservices.*)

The SDK provides classes for both input objects as output objects. These classes can be found in inputObjects and outputObjects packages of the xperido-client.jar respectively. Additionally, the packages jsonHelpers and xmlHelpers provide classes to convert between JSON strings, JSON objects, XML strings and POJO. Using these classes for custom objects not found in the inputObjects and outputObjects packages likely requires additional JAXB knowledge to make the conversions work properly.

7.4.3 Error handling

A webservice can fail to return a valid response for numerous reasons. We can divide these into three categories: input errors, server errors, and bugs in the SDK itself.

7.4.3.1 Input errors

Input errors occur when a developer gives faulty input to one of the methods in the XperidoNucleusServices. The most basic mistakes such as giving a wrong input type, a wrong amount of elements in an input array, or null instead of an effective object are detected before the webservice is invoked. These will result in runtime exceptions with a clear message of what went wrong. We implemented this exception system because you, as a developer have to be aware of what went wrong so you can mend the faulty code.

7.4.3.2 Server errors

Server errors relate mainly to establishing a connection and sending data over the connection. When they occur, there is nothing you as a developer can do to squeeze out a valid response anyway. Best practice would be to surround every call to a method that invokes a webservice with a try/catch block. If any exception is caught in this manner, that is a sign that no valid response could be fetched. Your application can then handle this problem in different ways: try again after some time, display an error message to a user, etc.

We did not implement an extensive exception system for server errors, because there is really nothing your code can do to prevent, for example, an internal server error. If this kind of error occurs and persists, please contact your XperiDo partner - they can check what went wrong with the server and find a solution.

7.4.3.3 Bugs

Bugs in the SDK can manifest when it runs out of insect spray. If you spy one, please contact your XperiDo partner. They will get right on it to provide you with a fresh can.

7.5 C# SDK

A .dll library file is available which contains all the necessary classes to invoke the XperiDo Nucleus webservices in an object-oriented way. To use it, add a system reference to this library in your project and instantiate the XperiDoNucleusServices class found in the XperiDoNucleusRestServicesAPI.services namespace. The constructor of this class requires parameters, but each parameter is fully documented. This class provides public methods for each web service. Developers can choose the type of response they receive from the web service by calling the appropiate methods. for example, the listtemplates webservice can be invoked by calling either of the following 6 methods:

1) public String getAllTemplatesAsJSON() - output: a JSON string.

2) public String getAllTemplatesAsXML() - output: an XML string.
3) public ListTemplatesResponse getAllTemplates() - output: a ListTemplatesResponse object.
4) public String getTemplatesByMarkerQueryAsJSON(Expression expression) - input: An Expression object, output: a JSON string.
5) public String getTemplatesByMarkerQueryAsXML(Expression expression) - input: An Expression object, output: an XML string.
6) public ListTemplatesResponse getTemplatesByMarkerQuery(Expression expression) - input: An Expression object, output: a ListTemplatesResponse object.

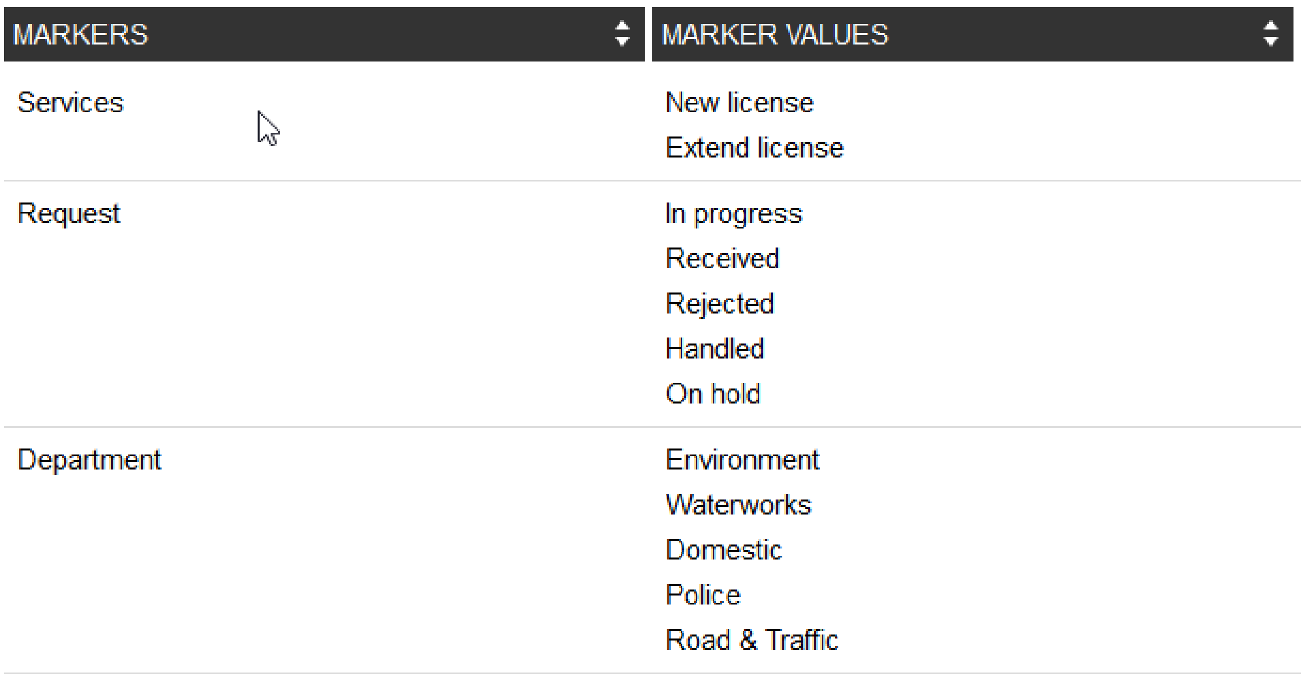
(*Note: the names of the methods in the XperiDoNucleusServices class are chosen to reflect what they do. They are not exactly equal to the names of the webservices themself, but it is easy to identify which methods(s) invoke which webservices.*)

The SDK provides classes for both input objects as output objects. These classes can be found in XperiDoNucleusRestServicesAPI.inputObjects and XperiDoNucleusRestServicesAPI.outputObjects namespaces of the .jar respectively. Additionally, the namespaces XperiDoNucleusRestServicesAPI.jsonHelper and XperiDoNucleusRestServicesAPI.xmlHelpers provide classes to convert between JSON strings, JSON objects, XML strings and POJO. Using these classes for custom objects not found in the XperiDoNucleusRestServicesAPI.inputObjects and XperiDoNucleusRestServicesAPI.outputObjects packages likely requires JAXB knowledge to make the conversions work properly.

8 Managing markers in the XperiDo Nucleus Web interface

8.1 Markers and marker values

Markers enable users to categorise templates. A marker has a name and an unlimited amount of marker values. A marker value has just a name. Each template can be associated with any amount of marker-marker value pairs. In the following example there are 3 markers: Services, Request and Department



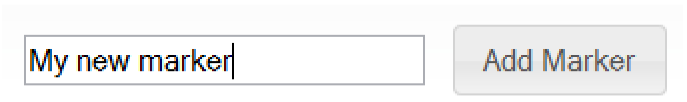
MARKERS	MARKER VALUES
Services	New license Extend license
Request	In progress Received Rejected Handled On hold
Department	Environment Waterworks Domestic Police Road & Traffic

Imagine a city has templates which contain a letter to inform a department that their new license request was succesfully received and they should contact the licence manager for their department. For each each department, the content of this template might be slightly different resulting in a different template for each department, for each request, and for each service. This amount of templates could quickly become unwieldy! If templates are marked accordingly, a filter can be applied to the listtemplates web service to only return templates matching the given filter. For example: *give me only the templates with 'Department - Domestic' and 'Request - Received'*.

Some common operations to create and manage markers and their marker values are described below:

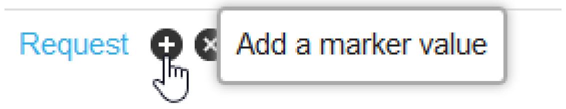
Adding a new Marker

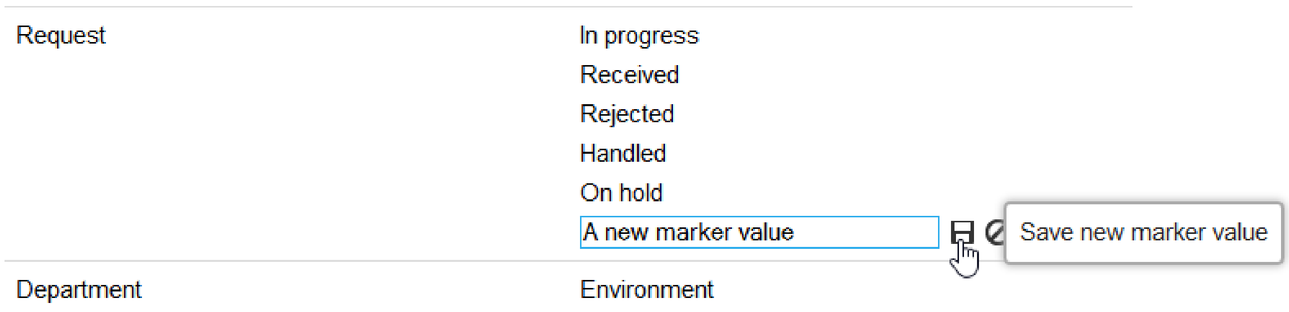
Type an appropiate name in the textfield on the top of the page and click on the button next to it that says "Add Marker".



Adding a new marker value

1) Hover your mouse over the marker you want to add a marker value to.
2) Click on the "+" button that appaers on the right off the marker's name.
3) A textfield will appear in the marker value column where you can type a name for the new marker value.
4) Click on the "save" button next to that textfield to add the new marker value to the marker.



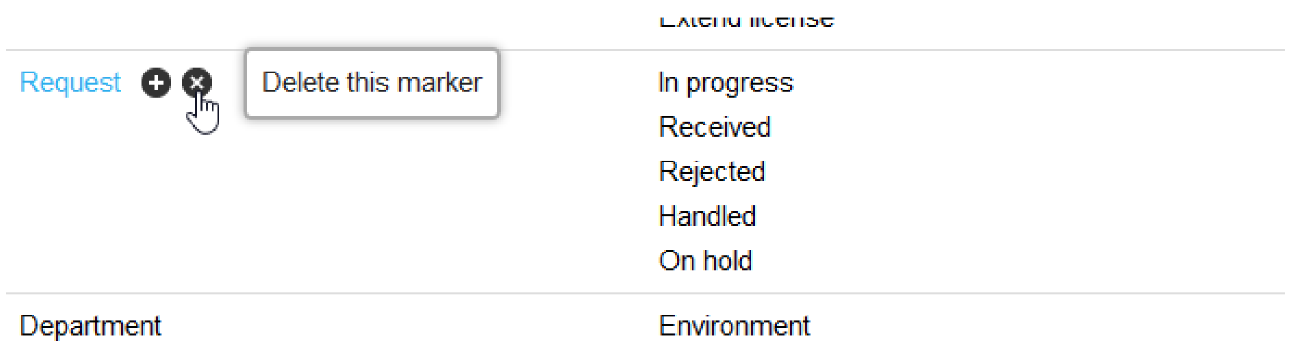


Request	In progress
	Received
	Rejected
	Handled
	On hold
	A new marker value
Department	Environment

Deleting a marker

1) Hover your mouse over the marker you want to delete.
2) Click on the "x" button that appears on the right off the marker's name.
3) A confirmation dialog appears. Click on "Delete" to permanently delete the marker.

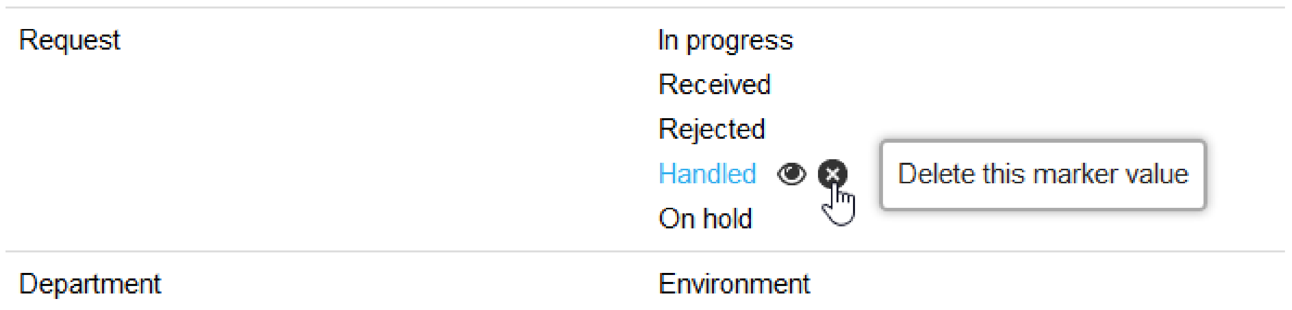
Deleting a marker will also automatically remove all its marker values, and remove those values from all templates.



Deleting a marker value

1) Hover your mouse over the marker value you want to delete.
2) Click on the "x" button that appears on the right off the marker value's name.
3) A confirmation dialog appears. Click on "Delete" to permanently delete the marker value.

Deleting a marker value will also automatically remove that value from all templates.



8.2 Adding markers and marker values to templates

9 Managing Dynamic Fields in the XperiDo Nucleus Webportal

9.1 Dynamic Fields

When a document is created from a template without dynamic fields, only data from predefined data set(s) can used to populate the document with values. Dynamic fields allow a template to not only use the "static" data from data sets, but also additional "dynamic" data that exists only during the document creation process. This dynamic data is given by the user who is creating a document, and it can be different each time the document is created. This is achieved by first defining which dynamic fields should exist for any given template (by creating dynamic field definitions), and then asking the user a value for each of these dynamic fields during the document creation process.

A **dynamic field definition** bundles all necessary information (called attributes) about a dynamic field, such as which type the dynamic field is, its name, its ID and other type-specific attributes. Each template can have an unlimited amount of dynamic field definitions associated with them. Dynamic field definitions are not sharable between different templates. This means if you create a dynamic field definition for one template, no other template will have access to it. Because it can be a burden to define equivalent definitions over and over again for different templates, we are currently developing a "copy" function where you can copy the dynamic field definitions from a template to another template. For now, you will have to manually create all the dynamic field definitions.

9.2 Adding and editing dynamic field definitions to a template

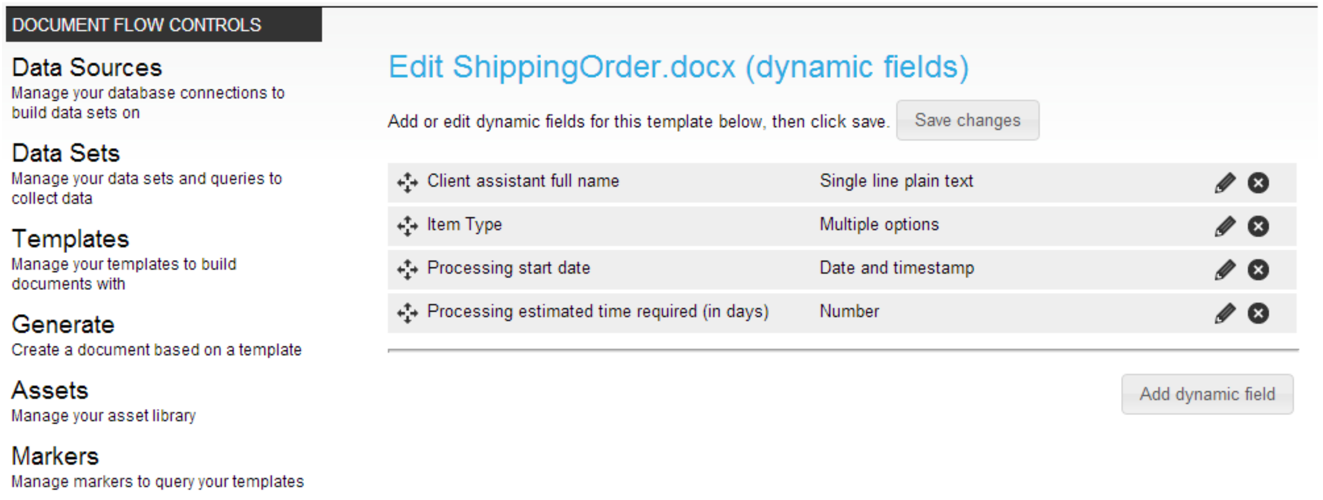
To add or edit the dynamic field definitions of a template, first navigate to the dynamic fields page on the XperiDo Nucleus Webportal:

1) Click on "Templates" in the leftsided navigation menu.
2) Click the dynamic field icon on the right of the name of the template you wish to edit.

An alternative way to set dynamic field definitions is

1) Click on "Templates" in the leftsided navigation menu.
2) Click on the name of the template you wish to edit.
3) When the template details page opens, click on the "Edit dynamic fields" button.

The dynamic fields page consists of two parts. On top is the list of earlier defined dynamic fields, and on the bottom is the panel to add or edit dynamic field definitions (this part is only visible when actually adding or editing).


DOCUMENT FLOW CONTROLS

Data Sources
Manage your database connections to build data sets on

Data Sets
Manage your data sets and queries to collect data

Templates
Manage your templates to build documents with

Generate
Create a document based on a template

Assets
Manage your asset library

Markers
Manage markers to query your templates

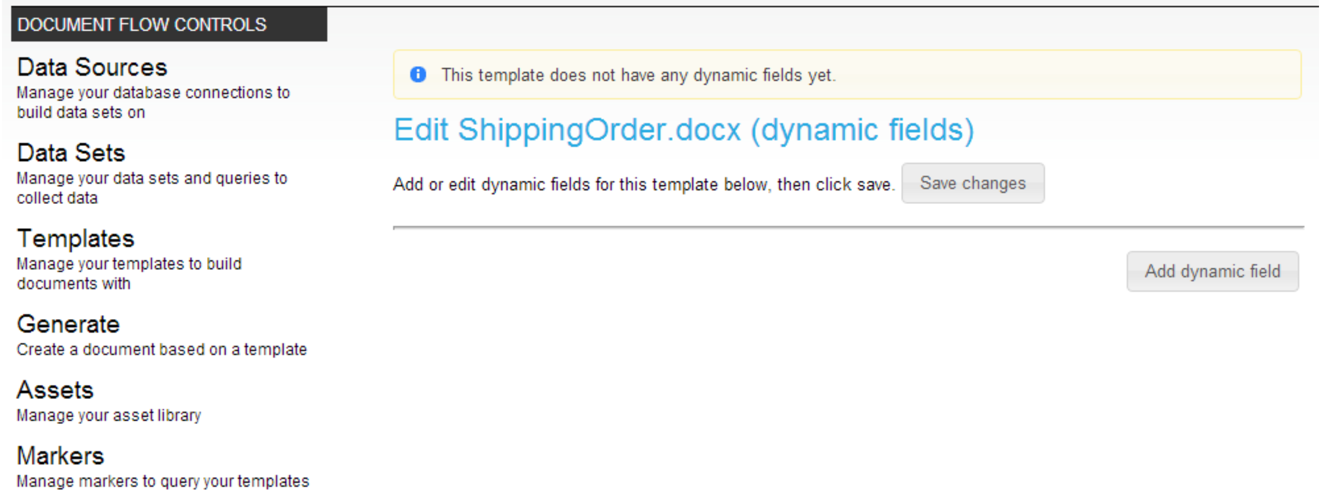
Edit ShippingOrder.docx (dynamic fields)

Add or edit dynamic fields for this template below, then click save. Save changes

Client assistant full name	Single line plain text
Item Type	Multiple options
Processing start date	Date and timestamp
Processing estimated time required (in days)	Number

Add dynamic field

It is possible a template has no dynamic fields set. In this case, a message will appear on top of the page to inform the user and the list of earlier defined templates will be empty.


DOCUMENT FLOW CONTROLS

Data Sources
Manage your database connections to build data sets on

Data Sets
Manage your data sets and queries to collect data

Templates
Manage your templates to build documents with

Generate
Create a document based on a template

Assets
Manage your asset library

Markers
Manage markers to query your templates

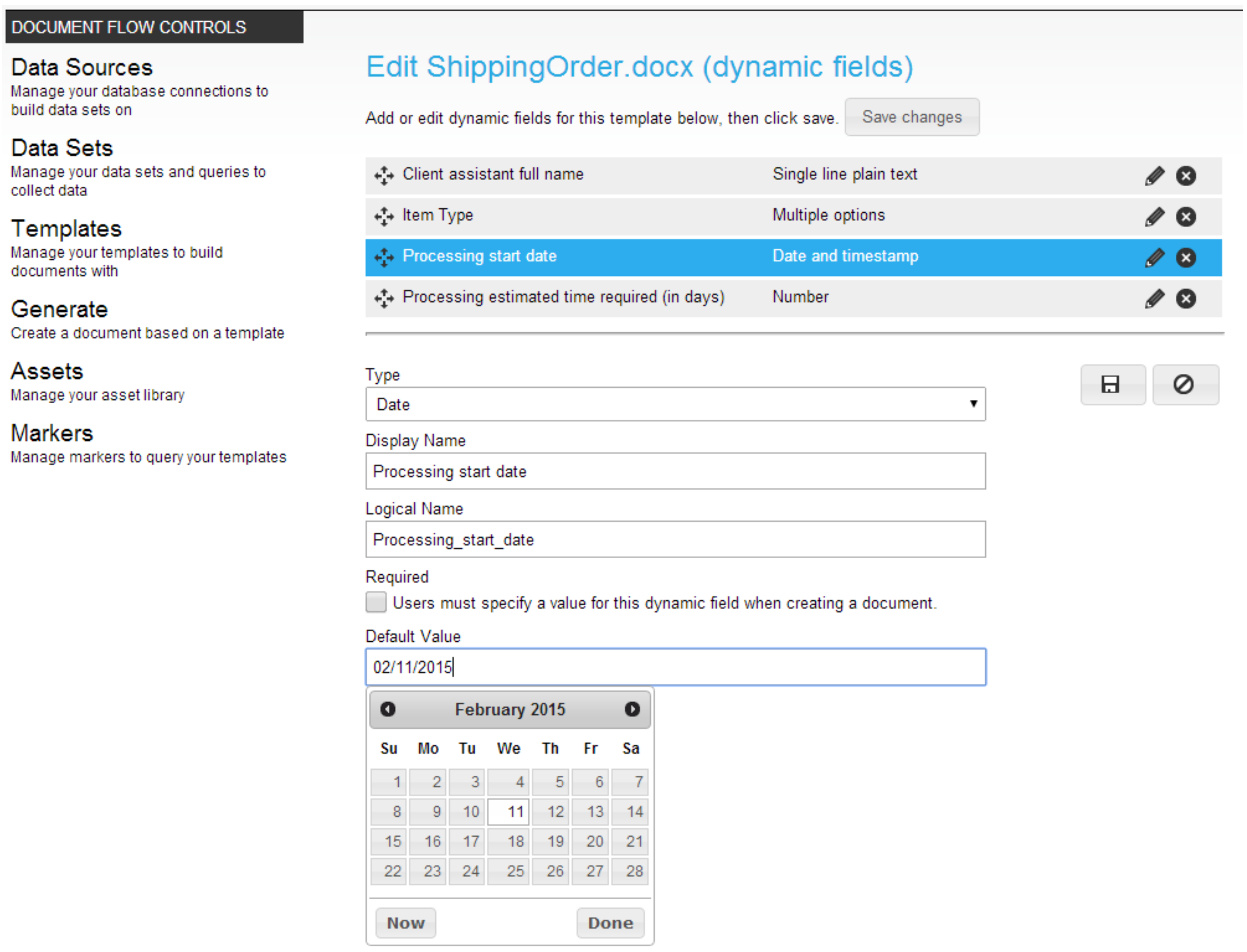
This template does not have any dynamic fields yet.

Edit ShippingOrder.docx (dynamic fields)

Add or edit dynamic fields for this template below, then click save. Save changes

Add dynamic field

Clicking on the "Add dynamic field" button on the bottom right will open the panel to create a new dynamic field definition. Similarly, clicking on the pencil icon on the right of an existing dynamic field definition will open the panel to edit that definition. While editing a dynamic field, it will be shown with a blue background in the list of dynamic fields instead of a grey background.



DOCUMENT FLOW CONTROLS

Data Sources
Manage your database connections to build data sets on

Data Sets
Manage your data sets and queries to collect data

Templates
Manage your templates to build documents with

Generate
Create a document based on a template

Assets
Manage your asset library

Markers
Manage markers to query your templates

Edit ShippingOrder.docx (dynamic fields)

Add or edit dynamic fields for this template below, then click save. Save changes

Client assistant full name	Single line plain text
Item Type	Multiple options
Processing start date	Date and timestamp
Processing estimated time required (in days)	Number

Type: Date

Display Name: Processing start date

Logical Name: Processing_start_date

Required
☐ Users must specify a value for this dynamic field when creating a document.

Default Value: 02/11/2015

February 2015

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

Now Done

It is important to press the "Save changes" button after defining the dynamic fields (unless the intent is to not save any additions or changes just made). Simple adding a new dynamic field definition will NOT save it permanently!

When adding a new dynamic field definition, it is appended to the bottom of the list of existing definitions. This list can be reordered freely by dragging the items to another position inside the list. To do this, left click on the dynamic field in the list, and drag it to the desired positon. The order of the definitions represents the order in which a user should fill in the values of the dynamic fields during the document creation process. In most situations, this ordering will not matter. However, this can be useful if asking for a value of a dynamic field would make no sense to a user who hasn't first specified previous values. As an example, imagine a user having to specify an "amount" first, and a "product ID" afterwards. it could be confusing to specify an amount if you don't know you're being asked the amount of a specific product.

9.3 Dynamic field types

There are seven types of dynamic fields:

1) Single line plain text
2) Multiline HTML Text
3) Number
4) Multiple options
5) Date
6) Timestamp
7) Date and timestamp

Before elaborating on each type individually, we will first discuss the attributes that each type of dynamic field has.

9.3.1 *Shared attributes of all dynamic field definitions*

- Type: This attribute must be set, and it can only be one of the seven types in the list above.
- Id: A unique GUID to identify this dynamic field. This id is automatically generated and is not visible in the XperiDo Nucleus Webportal. It is used mainly for internal processing. No other dynamic field (for this template) will have the same id.
- Logical name: A unique name to identify this dynamic field. It must be filled in when creating a new dynamic field. It can only contain letters (accented characters are allowed), digits and underscores. No other dynamic field for this template can have the same logical name.
- Display Name: A unique name to identify this dynamic field. It must be filled in when creating a new dynamic field. It can contain any character except <, >, &, ", '. No other dynamic field for this template can have the same display name.
- Default value: The value this dynamic field will have if the user does not explicitly specify a value. This attribute is optional.
- Required: A flag to mark that this dynamic field must have a value assigned to it during the document creation process. If this not checked, the user can choose not to fill in a value. If this is checked, the user MUST fill in a value, but the user can choose the default value if desired (if one is set).

9.3.2 *Single line plain text*

This field has no type specific attributes. The default value can be, like the name suggests, a single line of plain text.

The purpose of this dynamic field is to insert a few words or small sentences into your template. Imagine a template where the user creating the document can add a small personal note at the end. This note can be different for each user or each time the document is created. A default value could be "Sincerely, the technical team", but users could enter a new value if they want such as "Kind regards, John Doe".

9.3.3 *Multiline HTML text*

This field has no type specific attributes. The default value can be anything from a single letter to several paragraphs.

Unlike the **single lin plain text**, this dynamic field is meant to insert larger snippets of text, complete with formatting such as bold, colored or centered text. A use case could be a template where the first paragraph is a welcome text, and each time the document is being created this welcome text should be personalized to the specific client or situation. Because the text cannot be standardized with plain CRM data, a multiline HTML text can provide a flexible, last-minute piece of text within a larger document.

9.3.4 Number

This field has 4 type specific attributes.

- Precision: This attribute must be set, and its default value is 0. This attribute denotes the maximum amount of allowed numbers behind the decimal mark. For example, if the precision is set for to 2, values can be entered with either 0, 1 or 2 numbers behind the decimal mark. A value of zero thus indicates only integer values are allowed. This value cannot be negative (While this could be interpreted as a mandatory amount of 0's to be place in front of the decimal mark, it was chosen this is not worth the possible confusion).
- Minimum: The (default) value cannot be lower than this amount. This minimum must also obey the chosen precision. If a decimal mark should be present, it must be entered with a . (dot), not with a , (comma)!
- Maximum: The (default) value cannot be higher than this amount. This maximum must also obey the chosen precision. If a decimal mark should be present, it must be entered with a . (dot), not with a , (comma)!

Note that the default value is restricted by these attributes. If a minimum of 50 is specified, then the default value must be at least 50 as well.

Although there exists a dynamic field of type date, numbers can be used in case a specific year has to be filled in. With attributes precision = 0, minimum = 2008 and maximum = 2015, the user creating a document will be able to fill in any number (year) from the following list: 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015. Fractions are not allowed in this case because the precision is set to 0 (zero numbers behind the decimal mark).

If precision is set to a very high number, rounding errors could occur (because a computer cannot process all possible numbers). If this poses a problem, adjust your unit (for example, change kilometres to millimeters) to reduce the precision.

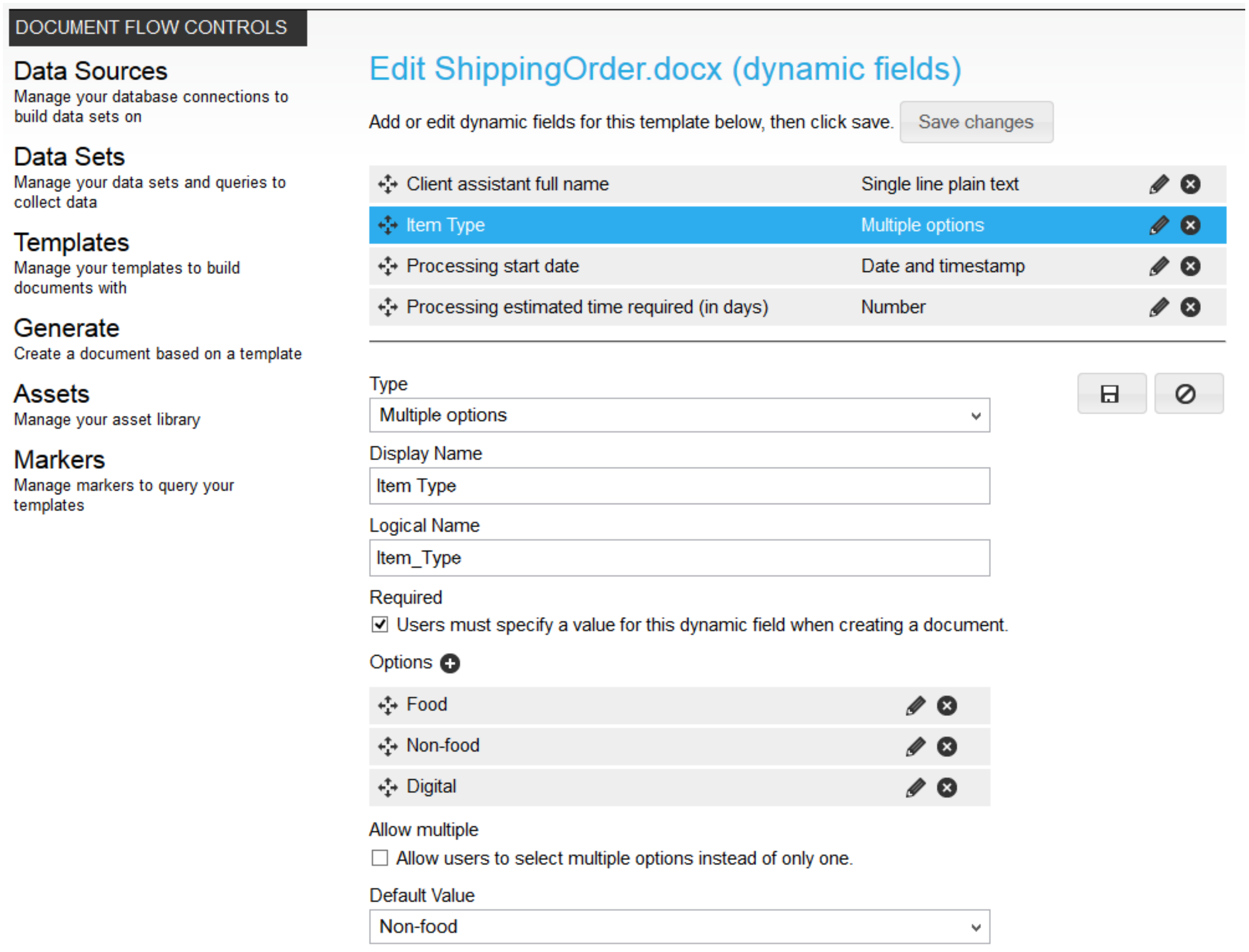
9.3.5 Multiple options

This field has 2 type specific attributes.

- Options: A list of possible options to choose from. An "option" is nothing more than a string. An example of this attribute would be the following collection of colors: ["red", "green", "yellow", "purple"] . by default, there are 2 options named "Option 1" and "Option 2". Options can be created by clicking on the + icon, and they can be deleted by clicking the x icon.Options can be renamed at any time, and the order of the options in the list can be changed by dragging the options to another position. No two options can share the same name.
- AllowMultiple: This attribute is either false or true. When it's true, more than one value can be selected by the user, and more than one default value can be chosen when creating a dynamic field definition.

Note that this dynamic field can also be used to ask the user a simple yes/no question by defining only the two options "Yes" and "No".

An example of a dynamic field of this type is shown below (in the screenshot, a user is editing the dynamic field definition "Item type").


DOCUMENT FLOW CONTROLS
Data Sources
Manage your database connections to build data sets on
Data Sets
Manage your data sets and queries to collect data
Templates
Manage your templates to build documents with
Generate
Create a document based on a template
Assets
Manage your asset library
Markers
Manage markers to query your templates
Save changes

Edit ShippingOrder.docx (dynamic fields)

Add or edit dynamic fields for this template below, then click save.

Client assistant full name	Single line plain text
Item Type	Multiple options
Processing start date	Date and timestamp
Processing estimated time required (in days)	Number

Type: Multiple options

Display Name: Item Type

Logical Name: Item_Type

Required
☑ Users must specify a value for this dynamic field when creating a document.

Options
- Food
- Non-food
- Digital

Allow multiple
☐ Allow users to select multiple options instead of only one.

Default Value: Non-food

9.3.6 Date

This field has no type specific attributes. The default value must be a date, formatted in "mm/dd/yyyy" format. To assist in typing a correct date, a datepicker widget will appear as soon as a value can be entered.

9.3.7 Timestamp

This field has no type specific attributes. The default value must be a timestamp, formatted in "hh:mm:ss" format, where hours respect the 24-hour system. To assist in typing a correct time, a timepicker widget will appear as soon as a value can be entered.

9.3.8 Date and timestamp

This field has no type specific attributes. The default value must be a date, formatted in "mm/dd/yyyy" format, followed by a timestamp, formatted in "hh:mm:ss" format, where hours respect the 24-hour system. To assist in typing a correct date and time, a date- and timepicker widget will appear as soon as a value can be entered.

10 Parameters for the create document service

Generating a document based on a data source

replyFormat

The type of the response. Can be either {XML} or {JSON}. This parameter optional, if it is omitted the default value {JSON } will be used.

template

The name of the template, for example: {my_template_5.docx} or {multipart_template.dmt}. The file extension can be omitted, but it is recommended to use so the repository look-up is faster. The extension {.docx} is used for non-multipart templates, and {.dmt} is used for multipart templates. This parameter must be set.

templateItemType

The item type of the template. This is either {TemplateArchive} or {DocMergeTemplate}. This should not be confused with template types as used in the XperiDo Nucleus web interface (such as Body, Header, Footer…). This parameter is optional unless a name clash occurs, for example: both {template1.xta} as {template1.dmt} exist and the {template} parameter value is {template1}, without the file extension.

query

The name of a query that is defined on the data set of the given template (As specified by the {template} parameter) Queries are created using the XperiDo Nucleus web interface. All data sets have one default query that doesn't need to be created, with name "all", which will use all the records in that data set. This parameter is optional, in which case the "all" query will be used.

key

A concatenation of all variable values used by the query as specified by the {queryName} parameter. The format for this parameter is the variable names separated by a semicolon {;}. for example: {"5;2014;Invoice"} on a query that has three variables. The order of the variable values must be the same as the order they were defined in the query. If the specified query has no variables (such as the default query with name "all"), then this parameter is optional. Otherwise this must be set.

language

If multiple languages are defined, this parameter indicates which language the created document must have. To know which languages are defined, the {listLanguages} service can be called which returns a list of string values. This parameter must be one of the returned strings (case sensitive). This parameter is optional, in which case the default language will be chosen. If only one language is defined in the project that language is automatically the default language. If the project has no languages, this parameter is ignored.

variant (not yet implemented!)

Each template can have different variants. These variants can be different for each template (unlike languages which are the same for each template). This parameter indicates which variant of the template the created document should be based on. To know which variants are defined, the {listVariants} service can be called which returns a list of string values. This parameter must be one of those strings (case sensitive) to choose a variant. However, this parameter is optional, in which case the default variant will be chosen. If only one variant exists, that variant is automatically the default version.

saveLocation

Any valid file path on the server where files can be written and read from. The path must be absolute, for example: {"C:/MyDirectory/XperiDoDocuments"}. If the directory does not exist, it will not be created and the document can not be saved. This parameter must be set.

saveName

The name of the generated document. It is allowed, but not necessary that this is the same name as the template name (with or without {.docx} or {.dmt} extension). The name can only contain characters that the server allows in file names. This parameter must be set.

saveNamingMode

A dynamically created suffix can be appended to the file name (as specified by the {saveName} parameter). This can be either the current date, the current date and time, or a sequential number (for example, if multiple documents share the same file name). This parameter is optional and can be null. In that case, no suffix will be added to the file name unless another document already has the same name, in which case a sequential number will be appended to resolve the name conflict.

saveFormat

The file extension of the generated document. This parameter must be set and can be any of these values: Bmp, Doc, Docm, Docx, Dot, Dotm, Dotx, Emf, Epub, FlatOpc, Html, Jpeg, Mhtml, Odt, Ott, Pdf, PdfA1b, Png, Rtf, Text, WordMl, XamlFlow, Xps

mailTo

The e-mail address to send the generated document too. This parameter is optional, in which case no e-mail will be send. E-mailing can be used in conjunction with printing the generated document. There is no confirmation of successful or failed of delivery. The server must be configured to allow these mailjobs. By default, this is not allowed.

mailSubject

The subject line of the send e-mail. This parameter is optional. It is ignored if no {mailToAddress} is specified.

printPrinter

The name of the printer the generated document will be printed on. To get the available printer names the server can print on, the {listPrinters} service can be used. This parameter is optional, in which case nothing will be printed. printing can be used in conjunction with e-mailing the generated document. There is no confirmation of successful or failed printing.

printCopies

The amount of copies to print. This parameter is optional. It is ignored if no {printerName} is specified. If this parameter cannot be parsed to a strictly positive integer, no printing will occur.

Generating a document based on an xsd.

Uses the same parameters as above, except it has no **query** or **key** parameter.